VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekce beztexturových 3D objektů

Detection of Textureless 3-D Objects

2017 Bc. Petra Svobodová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Petra Svobodová**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Detekce beztexturových 3D objektů
Detection of Textureless 3-D Objects

Jazyk vypracování: čeština

Zásady pro vypracování:

Sledování objektů je důležitou součástí počítačového vidění. Cílem práce je implementace algoritmu pro rychlou detekci beztexturovaných 3D objektů v obraze. Hledaný objekt je definován jako sada obrazů, které vyobrazují zadaný objekt pod různými úhly pohledu (pózami objektu). Ve výsledném obraze je pak objekt označen a s přiřazenou pravděpodobností k nalezenému vzoru.

Ve své práci proveďte:
1. Nastudujte aktuální stav poznání ohledně detekce beztexturovaných objektů.
2. Naimplementujte algoritmus [1] pro detekci objektů ve scéně. Výstupem bude též určení pózy hledaných objektů.
3. Svou implementaci řádně otestujte a zdokumentujte.
4. Proveďte závěrečné shrnutí.

Seznam doporučené odborné literatury:

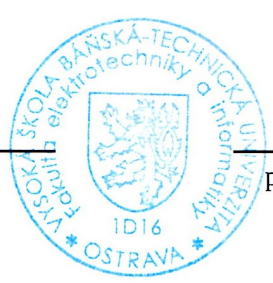
[1] Cai, H., Werner, T., Matas, J.: Fast Detection of Multiple Textureless 3-D Objects, Computer Vision Systems: 9th International Conference, ICVS 2013, pp. 103-112, 10.1007/978-3-642-39402-7_11, (2013)

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Gaura, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 26. dubna 2017

Svobodová Petra

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 26. dubna 2017 ..... Svobodová Petra

Poděkování patří Ing. Janu Gaurovi, Ph.D. za odborné konzultace a vedení mé diplomové práce.

Abstrakt

V této diplomové práci se zabývám popisem a implementací algoritmu pro rychlou detekci beztexturových 3D objektů v obraze. Cílem je nejen detekce, ale i určení přibližné pozice objektu. Ten je v obraze zachycen v odlišných velikostech pod různými úhly pohledu. Výstupem algoritmu je označení objektu včetně určení polohy a hodnoty pravděpodobnosti k nalezenému vzoru.

Klíčová slova: openCV, detekce beztexturových objektů, příznaky, indexovací tabulka, kaskádový klasifikátor, plovoucí okno, OCM, non-maxima suppression

Abstract

In this diploma thesis I deal with description and implementation of the algorithm for fast detection of textureless 3D objects in image. The target is not only detection, but also determine approximate position of the object. It is captured on the image by different sizes under different angles of view. Output of the algorithm is marked object with frame including positioning and value of probability to template.

Key Words: openCV, detection of textureless objects, features, index table, cascade classifier, scanning window, OCM, non-maxima suppression

Obsah

Seznam použitých zkratek a symbolů

OCM	–	Oriented Chamfer Matching
OpenCV	–	Open Source Computer Vision Library
CAD	–	Computer aided design

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Analýza a digitální zpracování obrazu se řadí mezi poměrně nové obory. Současná doba přináší mnoho problémů, pro které má analýza obrazu a počítačové vidění řešení. Neustále se vyvíjí nové algoritmy, které společně s moderními technologiemi usnadňují běžný život. S detekcí a rozpoznáváním objektů se setkáváme častěji, než by si člověk na první pohled pomyslel. S algoritmem pro detekci tváří se setkáme vždy, když náš fotoaparát automaticky zaostří na obličej před objektivem. Na parkovištích, kde mohou průmyslové kamery kontrolovat obsazenost jednotlivých míst či k rozpoznání SPZ automobilu, aby se člověk vyhnul zdlouhavému prokazování identifikačním průkazem.

Ovšem žádný z uvedených algoritmů nemá 100% úspěšnost. Nedá se na plně spolehnout. K detekci objektů dochází často na základě barvy či jasu. Avšak ty se mohou společně se světelnými podmínkami v průběhu dne měnit. A co teprve objekty, které jsou bez textury, tudíž nejdou rozpoznat podle barvy.

Hlavním cílem mé práce je implementace a popis algoritmu, který je schopen detekovat beztexturové objekty ve scéně. Řešením je algoritmus pro detekci beztexturových objektů, který je založený na rozpoznávání hran. Jelikož hrany kopírují tvar objektu, tak je právě tvar základním rozlišovacím faktorem, který od sebe odlišuje různé objekty. K tomu se přidávají problémy jako je rychlost detekce nebo různé pozice a velikosti objektů ve scéně. Všechny tyto problémy jsem vzala v potaz a snažila se je co nejlépe vyřešit. V první kapitole své práce objasním hlavní problémy, s kterými se běžně v analýze obrazu, obzvláště při detekci objektů, setkáváme. Nastíním také řešení v podobě několika současných algoritmů, které se zabývají detekcí beztexturových objektů. Poté vysvětlím všechny důležité pojmy a představím princip, který jsem pro řešení svého algoritmu použila. Poslední kapitola se věnuje implementaci a shrnutí dosažených výsledků.

2 Současné přístupy k detekci beztexturových objektů

Z videosekvencí či digitálních snímků jsme schopni získat množství důležitých informací, na základě kterých umíme vyhodnotit různé situace. Od těch primitivních, jako je nalezení objektů ve tvaru kružnice, až po ty opravdu náročné, se kterými se setkáváme zejména v medicíně a bezpečnosti. Typickým příkladem je počítání chromozomů v buňkách organismu člověka. Výpočet bychom sice mohli provést ručně pod mikroskopem, ovšem celý proces by byl zdlouhavý, obtížný a často nepřesný. Oproti tomu analýza chromozomů počítačem je rychlá a přesná. Z toho plyne, že cílem analýzy obrazu je přiblížit se co nejvíce lidskému vidění. Schopnosti lidského oka a mozku jsou úzce propojeny, neboť oko obraz zachytí, nervová vlákna přenesou informaci do mozku, který ji analyzuje a vyhodnotí situaci. Teoreticky funguje na stejném principu i kamera, která přenáší skrz kabel obraz do počítače, kde dojde k analýze a rozpoznání okolí.

V digitálním světě chápeme obraz jako dvourozměrné pole nebo-li matici hodnot, kde každé políčko reprezentuje *pixel (picture element)*. Jak už název napovídá, jedná se o nejmenší bezrozměrný bod, který můžeme na displeji zobrazit. Ten je zastoupen konkrétní hodnotou barvy nebo jasu. Každý bod (pixel) jednoznačně identifikují souřadnice $[x, y]$. V analýze obrazu postupně procházíme všechny pixely obrazu a získáváme informace týkající se barev, odstínů nebo tvaru objektů.

Ve snímcích, kde se nachází objekty bez textury, jsou nejdůležitějším informativním prvkem jejich *hrany*. Ty jsou základem pro vytvoření příznaků a následných vektorů příznaků. Hrany jsou způsobené především různou hloubkou objektů nebo jejich zakřivením, které definují tvar. K detekci pak nejsou potřeba informace o barvě či jasu objektu. Jedinou dostačující informací pro funkčnost celého programu jsou hrany objektu.

Nezbytnou součástí každé detekce je trénovací množina. Tato sada snímků reprezentuje objekty nejen různých tvarů, barev a velikostí, ale především zachycuje objekty z různých úhlů pohledu. Velikost trénovací množiny pak závisí na konkrétním typu algoritmu. A právě v počtu trénovacích snímků se náš algoritmus odlišuje od běžných algoritmů pro detekci objektů, neboť většina z nich potřebuje pro trénování klasifikátoru více snímků, na kterých je objekt zachycen ve stejné pozici. Oproti tomu náš algoritmus potřebuje pro úspěšnou detekci pouze jeden trénovací snímek pro každou pozici objektu.

2.1 Současnost

Detekce objektů se řadí do kategorie počítačového vidění (computer vision). Tato oblast se věnuje nejen detekci, ale i lokalizaci, sledování a rozpoznávání objektů. Cílem každého detektoru je, aby se co nejvíce přiblížil dokonalosti. To je stav, kdy detektor rozpozná všechny objekty v obraze a zároveň neidentifikuje žádné falešné. Vzhledem ke světelným podmínkám a ostatním vlivům je téměř nemožné vytvořit detektor se 100% úspěšností. Přesto se technologie počítačového vidění stále více rozšiřují do reálného světa, kde s nimi běžně přicházíme do styku. S využitím počítačového vidění se setkáváme v medicíně u vyhodnocení dat z mikroskopu či RTG snímků.

Své uplatnění našlo i v armádě, zejména v oblasti obrany, kde jsme schopni detekovat nepřátelské vojáky či vozidla. Ovšem v současné době se o počítačovém vidění hovoří často ve spojitosti s autonomními vozidly, která jsou označovaná jako budoucnost automobilového průmyslu.

S detekcí beztexturových objektů se často setkáváme v průmyslových nebo robotických aplikacích, kdy potřebujeme rozpoznávat a sledovat kovové či ocelové výrobky, což jsou objekty bez textury. Schopnost rozpoznávat, ale i určit přesnou pozici objektu využíváme na montážních linkách, kde je primárním cílem správně spojit dvě části, které mají být svařeny dohromady. Úroveň průmyslových procesů by se značně zlepšila, pokud by byla detekce spolehlivá a proveditelná v reálném čase. Rozpoznání a lokalizace objektů značně usnadní manipulaci robotických přístrojů, které operují s výrobními kusy. Očekává se tedy, že současné detektory by mohly být efektivnější a přesnější. V současné době existuje několik metod, které se snaží o co možná nejpřesnější a nejrychlejší detekci a rozpoznávání objektů v obraze. My si nyní představíme a vysvětlíme obecný princip a konkrétně si popíšeme některé z nich. Základní metody pro rozpoznávání beztexturových objektů dělíme do tří kategorií.

1. První způsob je založen na rozdělení obrazu do malých buněk a spočítání histogramu orientovaných gradientů pro každou z nich. Ty jsou následně použity pro srovnání dvou rozdílných obrázků, kde detekují přítomnost shodných objektů. Nevýhodou takových metod je výpočetní složitost.

2. Další skupina metod se snaží vytvořit trojrozměrné modely objektů. Srovnání se pak provádí na základě porovnání modelů.

3. Třetí skupina metod je založena na principu chamfer matching. Ten byl představen již v roce 1977. Nejprve je potřeba vytvořit hranové obrázky dvou porovnávaných snímků a následně spočítat vzdálenosti mezi nejbližšími hranovými body obou obrazů. Průměrná vzdálenost se pak použije k vyhodnocení podobnosti. Jinak řečeno, pro každý pixel hrany z trénovacího snímku hledáme nejbližší pixel hrany rozpoznávaného snímku.

2.1.1 Histogram orientovaných gradientů

HOG nebo-li histogram orientovaných gradientů je metoda založená na hodnotách intenzity gradientů. V principu stačí, abychom obraz rozdělili na menší oblasti, tzv. buňky. Je nutné, aby měly všechny buňky stejnou velikost. Nejčastěji zachycují oblast o velikosti 8×8 pixelů. Tyto malé oblasti poté sloučíme do větších oblastí, tzv. bloků. Z toho vyplývá, že obraz se skládá z bloků, které jsou tvořeny skupinou buněk. Pro každou z těchto buněk vybereme jednu hodnotu, která udává směr gradientu a zároveň reprezentuje celou tuto oblast. Orientace hrany bude vždy v rozsahu $0 - 360^o$. A jelikož se blok skládá z několika buněk, tak je reprezentován několika hodnotami gradientu, které ve výsledku vytváří histogram pro celou oblast bloku. Histogram reprezentuje zastoupení jednotlivých hodnot gradientu v daném bloku. Ideální rozmezí pro rozdělení intervalů histogramu je 40^o.

Ovšem tato metoda není tak jednoduchá, jak by se na první pohled mohlo zdát. Nezbytnou součástí HOG metody je předzpracování obrazu, kdy je potřeba převést obraz do stupňů šedi, aplikovat na něj filtr pro odstranění šumu či vyrovnat hodnoty jasu a osvětlení v obraze. Poté můžeme přejít již k výše zmíněnému výpočtu gradientů a histogramů pro každý blok. Následně provedeme normalizaci histogramů, které použijeme pro vytvoření klasifikátoru.

2.1.2 Template matching

Template matching se řadí mezi nejjednodušší způsoby, jak vzájemně porovnat a rozpoznat dva snímky. Jedná se o metodu pro nalezení a lokalizaci malé šablony ve velkém obraze. Na jedné straně máme vzor a na druhé straně řadu snímků, kterou je potřeba se vzorem porovnat. To uděláme tak, že postupně posouváme šablonu skrz obraz a vyhodnocujeme podobnost. Existuje mnoho způsobů, jak podobnost vypočítat. Jedním z nich je porovnání hodnoty pixelu ve výsledném obraze a trénovacím snímků. Z toho vyplývá, že se příliš nesmí měnit osvětlení a jasové podmínky. Druhým způsobem je korelace pixelů ve výsledném obraze a trénovacím snímků. Nevýhodou je, že metoda není invariantní vůči rotaci ani změně měřítka.

2.1.3 Chamfer matching

Chamfer matching funguje podobně jako template matching na principu porovnávání dvou snímků. Ovšem s tím rozdílem, že porovnává šablonu s testovacími snímky na základě distanční mapy, což je obraz, ve kterém každá hodnota pixelu udává vzdálenost k nejbližšímu bodu hrany. Distanční mapu je možné vytvořit pouze z hranového snímku, proto je třeba na obraz nejprve aplikovat Cannyho hranový detektor nebo podobný filtr pro detekci hran.

Poté co z hranového obrázku zjistíme distanční mapu, přijde na řadu samotné porovnávání a to tak, že si z šablony vezmeme všechny souřadnice hranových pixelů a podíváme se, co se pod těmito souřadnicemi nachází v testovacím snímku. Následně všechny tyto vzdálenosti sečteme a vydělíme počtem hranových pixelů. Snímky považujeme za shodné, pokud je výsledný průměr nižší, než námi stanovená prahová hodnota.

Jelikož je část našeho algoritmu pro detekci beztexturových 3D objektů založena na principu chamfer matching, budeme se tomuto tématu více věnovat v dalších kapitolách.

3 Detekce objektů pomocí hran

Existuje spoustu způsobů s využitím mnoha metod, jak správně docílit detekce objektů. Pro detekci obličejů v reálném čase se používá detektor, který navrhli v roce 2001 Viola & Jones. Ti ve své práci představili nová vylepšení jako je zrychlení výpočtu Haarových příznaků pomocí integrálního obrazu, výběr nejvhodnějších příznaků algoritmem AdaBoost a v neposlední řadě kaskádový klasifikátor, který vytváří několik stupňů kaskády, kde se v každé fázi zbavíme nejméně podobných snímků. Tento algoritmus zmiňuji, neboť většina úspěšných současných algoritmů pro detekci objektů výchází z původního řešení tohoto algoritmu.

V této kapitole si nejprve objasníme pojem hrana a představíme metody, které se používají k jejich detekci. Poté se budeme zabývat našim rychlým detektorem beztexturových 3D objektů, který je založený na rozpoznávání hran. Ten objekty nejen detekuje, ale určí i jejich přibližnou pozici a hodnotu, která udává podobnost k nalezenému vzoru. To vše z jednoho trénovacího snímku na jednu pozici objektu. Na začátku každé detekce tedy máme množinu trénovacích snímků, kde každý z nich zachycuje objekt v jiné pozici. Proto jsme schopni ke každému snímku z plovoucího okna nalézt vzor z množiny trénovacích dat. Cílem algoritmu je, abychom se postupně zbavovali nejméně podobných snímků. Celý tento proces probíhá na základě dvoustupňového kaskádového modelu.

Jelikož je první stupeň kaskády založený na porovnávání příznaků, přiblížíme si také výpočet příznaků vzdálenosti a orientace. Následuje představení principu pro porovnávání dvou vektorů příznaků, který vychází z podobnosti histogramů. Porovnávat každý histogram skenovacího okna se všemi histogramy snímků z trénovací množiny je sice teoreticky správně, ovšem prakticky časově neproveditelné. Proto samotné porovnávání histogramů nahrazujeme indexovací tabulkou, mezi jejíž výhody patří rychlé vyhledávání. Tím se značně urychlí běh celého algoritmu. Na konci prvního stupně kaskády nám zbyde jen několik málo snímků vzniklých výřezem skenovacího okna, které projdou do druhé části kaskádového modelu. Druhá část je sice velmi přesná a skutečně nám vrátí nejpravděpodobnější kandidáty na hledané objekty, ovšem výpočetně je časově náročná. Proto je nutné, aby se do této fáze dostal pouze omezený počet snímků. A právě v této kapitole si rozebereme tuto časově náročnou metodu chamfer matchingu vylepšenou o výběr diskriminativních hran a kompenzaci vůči zkreslení u jednoduchých objektů. Navíc by měl celý tento proces být proveditelný v reálném čase.

3.1 Obraz ve stupních šedi

Člověk je neustále obklopen barvami. Proto se i přístroje jako kamera či fotoaparát snaží napodobit lidské oko. Tato zařízení vnímají okolní svět barev a jejich cílem je tuto barevnou informaci přenést do počítače pro další zpracování. Jelikož se jedná o strojový proces, je možné jej popsat pomocí modelu barevného vidění. Nejznámějším a nejčastěji používaným schématem je **RGB model**, který byl definovaný již v roce 1931. Ten je založený na míchání tří základních barev. R (red) - červená, G (green) - zelená a B (blue) - modrá. Každá ze tří barev je zastoupená hod-

notou v rozsahu $0 - 255$, přičemz 0 říká, že tato barevná složka není ve výsledné barvě pixelu obsažena vůbec. [1]

Stejně barevně jsou zachyceny naše vstupní snímky. Ovšem pro náš algoritmus jsou nevhodné, proto musíme tyto barevné snímky převést na obrazy ve stupních šedi. Hlavním důvodem je pozdější detekce hran, kdy potřebujeme na vstupu právě obraz ve stupních šedi. To znamená, že tři hodnoty RGB, nahradíme jedinou hodnotou v rozsahu $0-255$, kde 0 reprezentuje černou barvu a 255 bílou. Na první pohled by se mohlo zdát, že stačí hodnoty R + G + B sečíst. Ovšem to není možné. Jelikož je lidské oko různě citlivé na vnímání jednotlivých barev, musíme při sčítání ke každé barevné složce přiřadit určitou váhu. Konkrétní velikost jednotlivých vah je znázorněna v rovnici (1). Hodnota Y pak vyjadřuje výsledný jas, který udává "svítivost"pixelu.

$$Y = 0,299R + 0,587G + 0,114B \tag{1}$$

3.2 Detekce hran

Stejně jako lidské oko rozpoznává na základě změny jasu hranice objektu, tak na stejném principu od sebe odlišuje objekty počítač při zpracování obrazu. Nejprve bychom si měli vysvětlit co to hrana je, jak vzniká a jaké typy existují.

Hrana je místo, kde dochází k náhlé změně jasu. [2] Podle průběhu této změny rozlišujeme několik druhů hran – skokové, náběžné, impulsní (dvojice skokových hran) a střechové (dvojice náběžných hran). Jednotlivé typy hran jsou znázorněny na obrázku 1.

Obrázek 1: Typy hran - (zleva) skoková, nábežná, impulsní, střechová

Metody založené na detekci hran využívají faktu, že každý objekt v obraze je tvořen souvislou oblastí:

- Souvislá oblast je obklopena hranicí.
- Hranice je tvořena hranami.
- Hrany se skládají z jednotlivých pixelů (bodů hrany).

Hranu charakterizujeme velikostí a směrem $\varphi(x, y)$ gradientu. Víme, že směr hrany $\psi(x, y)$ je kolmý na směr gradientu. A za velikost hrany $e(x, y)$ považujeme velikost gradientu. Výpočet těchto veličin je znázorněn v rovnicích (2,3,4).

$$\varphi(x, y) = \arctan\left(\frac{f_y(x, y)}{f_x(x, y)}\right) \tag{2}$$

$$\psi(x, y) = \varphi(x, y) + \frac{\pi}{2} \tag{3}$$

$$e(x, y) = \sqrt{f_x^2(x, y) + f_y^2(x, y)} \tag{4}$$

Při teoretickém popisu předpokládáme, že se jedná o spojitý obraz a využíváme tedy parciální derivace, jejichž výpočet je znázorněn v rovnicích (5,6). Ovšem v praxi často pracujeme s diskrétním obrazem. Jak lze vidět v rovnicích (7,8), stačí v takovém případě nahradit derivace diferencemi a zbytek postupu zůstává stejný.

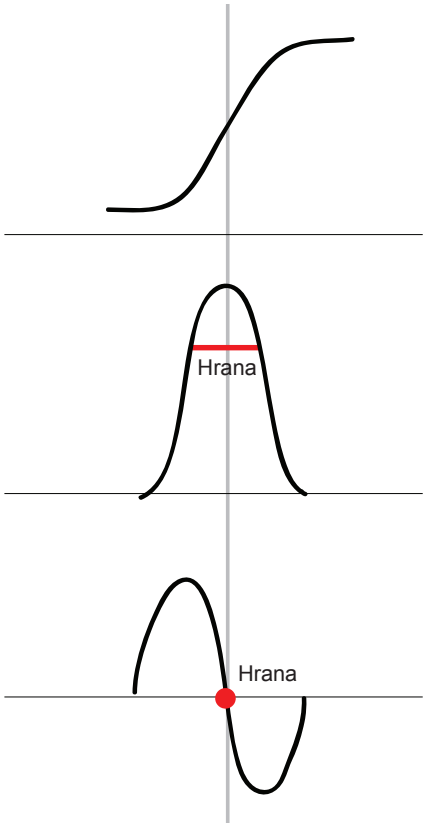
$$f_x(x, y) = \frac{\partial f(x, y)}{\partial x} \tag{5}$$

$$f_y(x, y) = \frac{\partial f(x, y)}{\partial y} \tag{6}$$

$$f_x(x, y) = f(x, y) - f(x - 1, y) \tag{7}$$

$$f_y(x, y) = f(x, y) - f(x, y - 1) \tag{8}$$

K detekci hran pomocí gradientu máme k dispozici několik základních technik – první derivaci, druhou derivaci a parametrické modely hran. Víme, že absolutní hodnota první derivace průběhu jasu má v místě hrany vysokou hodnotu. Našim cílem je tedy nalézt lokální maxima. Naopak v případě druhé derivace je v místě hrany hodnota rovna nule, neboť okolí hrany nabývá dvou opačných extrémů. [8] Oba tyto případy jsou vykresleny na obrázku 2.

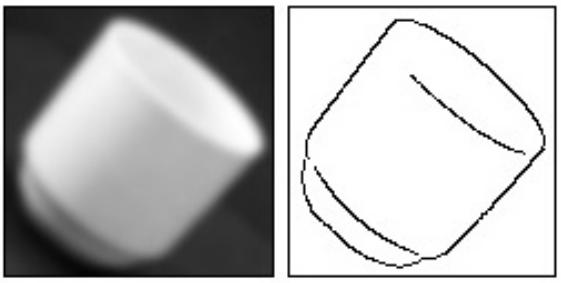


Obrázek 2: Hrana (nahoře) v průběhu první dervice (uprostřed) a druhé derivace (dole)

Metody používané k nalezení hran vychází z derivací a obecně se nazývají *hranovými operátory*. Mezi metody, které využívají první derivaci se řadí Cannyho, Sobelův, Robertsův, Prewittové a Kirschův operátor. Na druhé derivaci je založený Laplaceův hranový operátor. Většina těchto filtrů funguje na principu aproximace konvolučním jádrem a my si je teď blíže představíme.

3.2.1 Cannyho detektor hran

S detekcí hran je spojená celá řada problémů a v praxi jsou snímky často znehodnoceny šumem a pro detektor jsou pak jednotlivé hrany špatně čitelné. Můžou tak být rozpoznány nadbytečné hrany, které nenáleží objektu nebo naopak nebudou detekovány hrany, které by měly charakterizovat ohraničení objektu. To je důvod, proč je nejoptimálnější možností právě Cannyho detektor, který představil John F. Canny v roce 1986. Dosavadní algoritmy fungovaly spíše intuitivně, proto Canny stanovil základní požadavky ideálního hranového detektoru.[6] Ukázka vzorového objektu a jeho hranového obrazu, který vznikl aplikací Cannyho detektoru je znázorněna na obrázku 3.



Obrázek 3: Původní vzor a jeho hranový obraz

Mezi Cannyho nejdůležitější požadavky pro zlepšení detekce hran patří:

- Minimalizace chybné detekce.
- Přesné určení polohy hrany.
- Jednoznačná identifikace bodu hrany v obraze.

Princip výpočtu hran Cannyho detektorem:

1. K eliminaci šumu se využívá Gaussův filtr. K aplikaci Gaussova filtru použijeme konvoluci. Konvoluční maskou postupně vynásobíme všechny pixely obrazu. Tím docílíme jeho rozmazání a zbavíme se šumu. Princip je znázorněný v rovnici (9).

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{9}$$

2. Výpočet velikosti a orientace (směru) gradientu pomocí první derivace. Často se pro tento krok využívá Sobelův filtr.

3. Nalezení lokálních maxim. V tomto kroku odstraníme nevýznamné pixely a necháme jen ty, s nejvyšší hodnotou gradientu. Tím dojde ke ztenčení detekovaných hran a přesné identifikaci hranových pixelů.

4. Prahování s hysterezí. Výsledné pixely ještě musí projít dvojitým prahováním. Za body hrany jsou ozačeny ty pixely, jejichž hodnota je menší než horní hodnota prahu a zároveň ty, jejichž hodnota je větší než dolní hodnota prahu. Výsledek je odstranění malých nedůležitých hran.

3.2.2 Robertsův operátor

Přestože se jedná o jeden z nejstarších a nejjednodušších hranových operátorů, tak se používá dodnes i když ne v takové míře jako novější operátory. K aproximaci využívá konvoluční jádro 2×2. Výhodou je vysoká rychlost algoritmu. Oproti tomu je nevýhodou velká citlivost na okolní šum. Jelikož operátor pracuje s malou velikostí masky, tak často vyhodnotí šum jako hranu.

$$A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

3.2.3 Operátor Prewittové

Operátor Prewittové využívá hodnoty všech okolních pixelů. Pro konvoluci se používá jádro velikosti 3×3. Z toho plyne, že k výpočtu potřebuje všechny sousední hodnoty. Výsledná hodnota se poté vypočítá jako průměr.

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad B = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

3.2.4 Sobelův operátor

Sobelův operátor je podobný operátoru Prewittové. Gradient určujeme pomocí konvolučního jádra 3×3, takže potřebujeme k výpočtu všechny sousední pixely. Ovšem oproti předchozímu filtru, kde se pro výpočet používá aritmetický průměr, vylepšujeme Sobelův operátor o vážený průměr. Ideální pro detekci vodorovných či svislých hran.

$$A = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad B = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

3.2.5 Kirschův operátor

Kirschův operátor počítá derivaci ve všech osmi směrech. Jako výslednou hodnotu velikosti hrany bere maximální hodnotu.

$$A = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & 3 \\ -5 & -5 & -5 \end{bmatrix} \quad B = \begin{bmatrix} -5 & 3 & 3 \\ -5 & 0 & 3 \\ -5 & 3 & 3 \end{bmatrix}$$

3.2.6 Laplaceův operátor

Laplacián využívá druhé derivace změny průběhu jasu (viz rovnice (10)). Opět máme k dispozici konvoluční masku 3×3. Přestože maska počítá se všemi okolními sousedy, můžeme podle typu masky rozhodnout, zda výsledek ovlivní všech osm sousedů anebo jen čtyři z nich. Laplacián patří mezi metody citlivé na šum.

$$f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2} \tag{10}$$

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

3.3 Kaskáda klasifikátorů

Cílem našeho algoritmu je nalézt k snímku skenovacího okna, na kterém se nachází hledaný objekt, co možná nejpodobnější snímek z trénovací množiny dat, kterou tvoří šablony s vyhledávaným objektem. Postupně se zbavujeme méně podobných snímků, až dojdeme vyřazováním k nejpodobnějšímu z nich. Odborně se tento princip, kdy dojde v prvním kroku k zbavení většiny nepodstatných snímků nazývá kaskádový klasifikátor. Ve skutečnosti jde o to, že je výsledný klasifikátor složen z několika jednodušších klasifikátorů. Snímky tedy projdou několika fázemi, kdy se v každé z nich zbavíme nejméně podobných snímků.[4] Obecně říkáme, že kaskáda klasifikátorů se skládá z několika stupňů.

Základem našeho algoritmu je dvoustupňový kaskádový klasifikátor. Důležité je, aby došlo v prvním stupni kaskády, která není výpočetně tolik náročná, ke zbavení minimálně 90% všech potencionálně shodných snímků. Proto v druhém stupni kaskády, kde je výpočet přesnější, ale také časově náročnější, proběhne méně výpočtů a porovnání. Výsledný algoritmus tak poběží v kratším čase.

3.4 Příznaky (features)

Než přejdeme k samotné extrakci příznaků, tak předpokládáme, že jednotlivé objekty v obraze už jsou separovány. Tento krok je velmi důležitý, neboť *„příznakové metody rozpoznání jsou založeny na tom, že každý objekt, který má být rozpoznán, je popsán pomocí vhodných číselných hodnot, tzv. příznaků."*[6] Pokud by se v obraze nacházelo více neseparovaných objektů, postrádal by popis pomocí příznaků smysl.

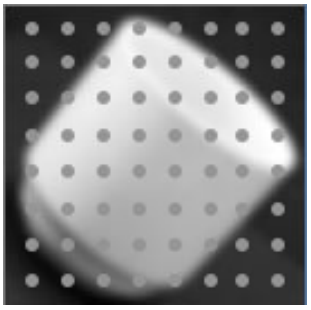
Jeden příznak pro popis každého objektu by mohl být matoucí, proto je ideální pro popis objektů použít více příznaků. Doporučuje se, aby jich bylo více než jeden a zároveň by jejich počet neměl překročit 10 - 20 vlastností. Jednotlivé příznaky x_i pak sdružujeme do vektoru příznaků $x = (x_1, x_2, \ldots, x_n)$.

Potíž nastává při volbě vhodných příznaků, jelikož jedinou informaci kterou budeme při porovnávání o objektu vědět jsou jednotlivé hodnoty příznaků. Jejich volba je tedy klíčová pro správnou detekci. Správný návod jaké příznaky zvolit neexistuje, ale doporučuje se volit takové příznaky, které od sebe dokážou objekty maximálně odlišit.

V minulosti se místo příznaků využívaly vlastnosti pixelů. Postup to byl sice správný, ovšem výpočetně náročný. Oproti tomu je srovnávání příznaků vyhodnoceno podstatně rychleji a přesněji.

3.4.1 Mřížka referenčních bodů

Zavedením bodů referenční mřížky se vyhneme tomu, abychom museli počítat příznaky pro všechny pixely vstupního obrazu. Pokud omezíme výpočet na skupinu pixelů, tak dojde ke zrychlení běhu celého programu, přestože zůstane funkčnost algoritmu zachovaná. V principu budeme stále popisovat celý obraz, jen zredukujeme počet hodnot ve vektoru příznaků.



Obrázek 4: Znázornění bodů referenční mřížky

Ve vstupním obraze je potřeba vytvořit skupinu bodů, na které se budeme v průběhu celého algoritmu odkazovat. Označíme je jako referenční body p. Body uspořádáme do mřížky se vzájemně pravidelným odstupem pixelů tak, aby referenční body pokrývaly celý obraz. Stejně jako na obrázku 4, kde šedé puntíky reprezentují body referenční mřížky. Poté co vybereme významné body, je potřeba k nim přiřadit příznaky (features). O to se postará deskriptor neboli popisovač. Ten na základě námi zvoleného postupu přiřadí k jednotlivým bodům konkrétní vlastnosti – příznaky.

V našem případě se ke každému referenčnímu bodu p vztahují dva příznaky – *vzdálenost* $d_i(p)$ k nejbližší hraně a *orientace* $\phi_i(p)$ této hrany. Z toho plyne, že referenční mřížku a výpočet příznaků je potřeba provést pro všechny vstupní obrázky. Nejprve pro trénovací a následně i pro testovací snímky. Vypočtené hodnoty pro vzdálenost a orientaci uložíme do vektoru příznaků. Vektor příznaků je vektor, jehož prvky jsou tvořeny jednotlivými hodnotami příznaků. První polovinu vektoru tvoří příznaky vzdálenosti všech bodů referenční mřížky a druhou polovinu tvoří příznaky orientace rovněž pro všechny body referenční mřížky. V našem případě bude vektor příznaků vypadat takto $(d_I(p_1), \ldots, d_I(p_m), \phi_I(p_1), \ldots, \phi_I(p_m))$, kde I je označení obrazu a m počet bodů referenční mřížky.

3.5 Výpočet příznaku vzdálenosti

Přepokladem pro úspěšný výpočet vzdálenosti (distance) dvou pixelů je vstupní obraz v binárním tvaru. *Binární obraz* je takový obraz, ve kterém se vyskytují pouze dvě barvy – bílá a černá. Obecný způsob jak dosáhnout takového obrazu se nazývá *prahování (thresholding)*. Další podmínkou pro výpočet vzdálenosti je, aby se jednalo o obraz, ve kterém jsou vyznačené pouze hrany objektu. K nalezení hran a vytvoření binárního obrazu nám poslouží *hranový deskriptor*.

Před samotnou hranovou detekcí musíme obraz převést do stupňů šedé. Výstupem hranového deskriptoru by měla být hrana, která ohraničuje jednotlivé objekty. *„Za bod hrany se nejčastěji považuje místo, kde průběh jasu vykazuje náhlou změnu.“*[5] Z toho plyne, že každý pixel, ve kterém dochází k přechodu jasu je významným pixelem, který tvoří hranu či hranici obklopující objekt. Ovšem nalezení hran nebývá jednoduchý problém. Jak jsme si ukázali výše, existuje několik osvědčených metod pro detekci hran. Jednotlivé kroky jsou velmi podobné. Postupně se projdou všechny pixely $[x, y]$ vstupního obrazu a pro každý z nich se provede specifická operace. Nejčastěji se jedná o konvoluci s vhodným konvolučním jádrem. Náš algoritmus je založený na Cannyho detektoru hran, který je ze všech dostupných hranových operátorů nejoptimálnější.

3.5.1 Distanční mapa

Samotný výpočet vzdálenosti proběhne na základě vzdálenostní transformace (distance transform). Aplikací transformace na náš vstupní obraz dostaneme *distanční mapu*. Jedná se o obraz, kde jsou jednotlivé hodnoty pixelů zastoupeny informací o vzdálenosti nejbližšího hranového pixelu. Z toho logicky vyplývá, že v místě hranového pixelu bude distanční hodnota rovna nule.

Jak vypadá distanční mapa můžeme vidět na obrázku 5. Z něj je patrné, že hodnoty jednotlivých pixelů reprezentují vzdálenost pixelů od hranice. Ta je nejen barevně odlišená, ale navíc jsou jednotlivé body hrany označeny číslem 0.

3	2	1	2	3	4
2	1	0	1	2	3
1	0	1	0	1	2
1	0	1	1	0	1
0	1	2	2	1	0
0	1	2	3	2	1

Obrázek 5: Distanční mapa

K dispozici máme několik metrik L_p, které slouží k výpočtu distanční mapy, tj. vzdálenosti mezi konkrétním pixelem a bodem nejbližší hrany. Cílem je tedy spočítat vzdálenost bodů $X[x_1, x_2]$ a $Y[y_1, y_2]$ v dvojrozměrném prostoru.[7]

- L_1 norma - Manhattanská vzdálenost (viz rovnice (11))

$$d_M(X, Y) = |x_1 - y_1| + |x_2 - y_2| \tag{11}$$

- L_2 norma - Euklidovská vzdálenost (viz rovnice (12))

$$d_E(X,Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \tag{12}$$

- L_∞ norma - Čebyševova vzdálenost (viz rovnice (13))

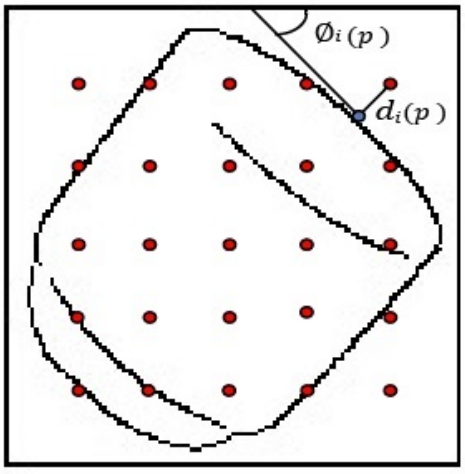
$$d_C(X,Y) = max(|x_1 - y_1|, |x_2 - y_2|) \tag{13}$$

Nejznámnější a nejčastěji používanou metrikou je bezesporu Euklidovská vzdálenost. V principu se vzdálenost počítá tak, jako by mezi dvěma porovnávanými body vedla přímka. Přesto jsem rozhodla ve svém algoritmu použít vzdálenost Manhattanskou, která je v našem případě ideální pro další kroky algoritmu.

3.6 Výpočet příznaku orientace

Druhým příznakem použitým pro popis snímků je orientace. Konkrétně orientace v bodě hrany, který je nebližší k bodu referenční mřížky. Nejprve je třeba zjistit souřadnice $[x, y]$ nejbližšího hranového pixelu pro každý bod referenční mřížky. K tomu využijeme distanční mapu získanou v předchozím kroku. Nejbližší hranový pixel nalezneme tak, že v místě bodu referenční mřížky zkontroluju jeho čtyři okolní sousedy – pravý, levý, horní a dolní pixe posunuté o distanční vzdálenostl. Ten s hodnotou rovnou nule je vítěz. To je totiž náš hledaný hranový pixel. Nyní známe výsledné souřadnice $[x, y]$. Teď už sice známe výsledny bod, ale ještě potřebujeme zjistit úhel orientace ψ, který svírá daná hrana s osou x.

Na obrázku 6 vidíme hranový snímek vzorového objektu hrnku, na kterém jsou červenými puntíky vyznačené body refereneční mřížky p_i. Vzdálenost mezi referenčním bodem a nejbližším bodem hrany (znázorněný jako modrý puntík) je označená $d_i(p)$. Od něj vede čára, která znázorněju směr hrany. $\theta_i p$ pak znázorňuje úhel, který reprezentuje orientaci hrany.



Obrázek 6: Znázornění výpočtu orientace

Vycházíme opět z hranového obrazu, kde použijeme pro výpočet orientace parciální či diferenciální derivaci. Hrana je místo s největší změnou intenzity. Naopak v místě, kde se hrana nenachází, je hodnota první derivace rovna nule, jelikož zde k žádné změně intenzity nedochází. Orientace hranových pixelů udává směr, který je reprezentován úhlem ψ (ve stupních nebo radianech). Při výpočtu postupujeme následovně:

1. Spočítáme derivace pro osy x a y. Pro tento výpočet využijeme buď klasickou derivaci nebo využijeme diference, kdy stačí hodnoty dosadit do vzorce pro centrální diferenci. [8]

$$\bigtriangledown f(x) = \frac{f_{+1} - f_{-1}}{2} \tag{14}$$

   Pomocí čtyř okolních pixelů bodu $[y, x]$ získáme diferenciální rovnice ve směru x a y:

$$\frac{\partial f}{\partial x}(y, x) = \frac{f(y, x+1) - f(y, x-1)}{2} \tag{15}$$

$$\frac{\partial f}{\partial y}(y, x) = \frac{f(y+1, x) - f(y-1, x)}{2} \tag{16}$$

2. Když známe hodnoty parciálních derivací, dosadíme je do funkce pro výpočet úhlu.

$$\psi = \arctan\left(\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}\right) \tag{17}$$

Takto spočítáme úhel orientace $\phi(p_i)$ pro každý bod referenční mřížky a vložíme jej do vektoru příznaků pro daný obraz I. Trénovací fáze spočívá v tom, že vytvoříme vektor příznaků pro každý vstupní trénovací snímek $(d_I(p_1), \ldots, d_I(p_m), \phi_I(p_1), \ldots, \phi_I(p_m))$. Poté si v testovací fázi vytvoříme vektor příznaků pro každý snímek posuvného okna, které postupně prochází celý testovací snímek $(d_T(p_1), \ldots, d_T(p_m), \phi_T(p_1), \ldots, \phi_T(p_m))$. V této fázi máme na jedné straně vektorové příznaky trénovacích snímků a na druhé straně vektorové příznaky testovacích snímků. Nyní můžeme vektory příznaků navzájem porovnat. Pokud nalezneme shodu nebo vysokou podobnost, tak jsme nalezli vzor a jeho obraz.

3.7 Indexovací tabulka

Jak už jsme zmínili výše, tak první stupeň klasifikátoru slouží ke zbavení až 90% zbytečných snímků posuvného okna. Tzn. snímků s nulovou nebo velmi nízkou podobností s vzorovými nebo-li trénovacími obrázky. O tom, zda jsou snímky shodné nebo ne, rozhodujeme na základě vyhledávání a srovnávání v indexovací tabulce.

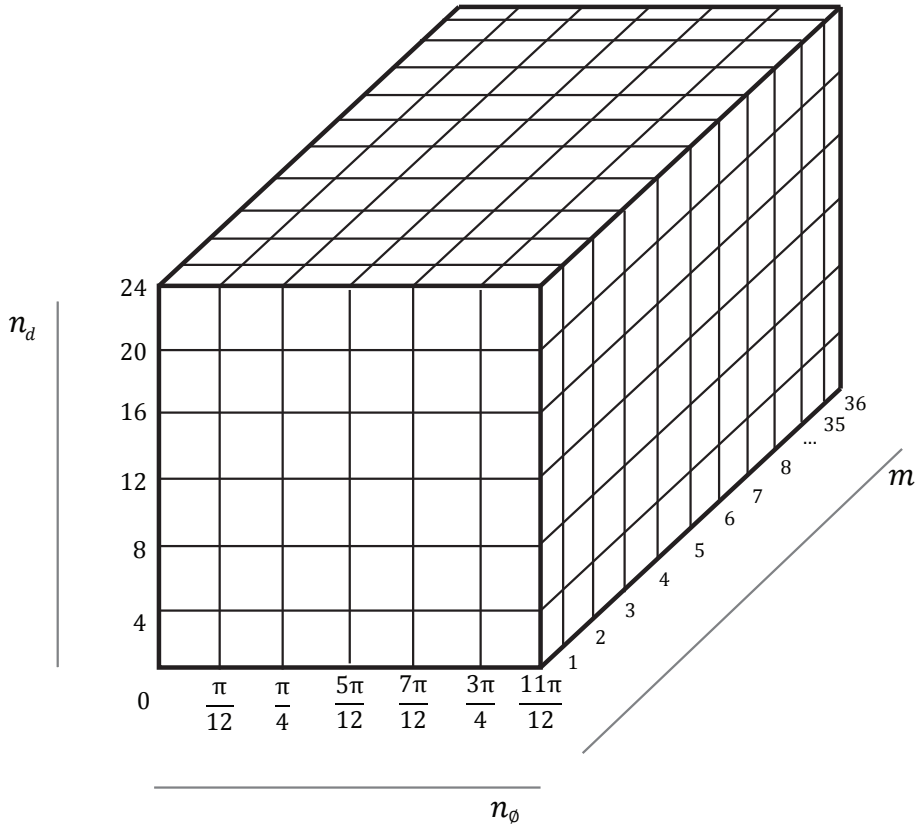
První stupeň kaskádového klasifikátoru je založený na porovnávání vektorů příznaků. Celý tento proces vychází z podobnosti histogramů. A proč je porovnávání založeno zrovna na podobnosti histogramů? Protože je nutné, abychom všechny příznaky vzdálenosti a orientace roztřídili

do konrétních intervalů. Stejně jako se přiřazují hodnoty do jednotlivých intervalů při tvorbě histogramů. Nejprve bychom si tedy měli vysvětlit co to histogram je a jak jej sestavit. *Histogram* je graf, který znázorňuje rozložení četností jednotlivých hodnot. Abychom zkonstruovali histogram, tak musíme nejprve data rozdělit do košů (binů), kde každý koš reprezentuje určitý interval. [9] No a úplně stejně postupujeme při tvorbě indexovací tabulky. Základem je kvantizace všech hodnot příznaků a rozdělení do jednotlivých košů. Distanční příznaky $d_T(p_i)$ a příznaky orientace $\phi_T(p_i)$ jsou kvantizovány do n_d a n_ϕ košů. Rozdíl mezi histogramem a tabulkou spočívá v tom, že histogram tvoříme na základě jednoho příznaku. Oproti tomu tabulka je zkonstruována na základě dvou příznaků. Tím pádem dostaneme tabulku o určitém počtu řádků (reprezentující hodnoty orientace) a sloupců (reprezentující hodnody vzdálenosti), kde se uvnitř buňky nachází hodnota četnosti.

Snímek skenovacího okna I je shodný s trénovacím snímkem T, jestliže má shodný dostatečný počet referenčních bodů p_i, tj. bodů referenční mřížky. Bod p_i považujeme za shodný právě tehdy, jestliže hodnoty $d_T(p_i)$, $d_I(p_i)$ a $\phi_T(p_i)$, $\phi_I(p_i)$ mají stejnou kvantizační hodnotu. Porovnávání jednotlivých hodnot příznaků je sice správné, ale časově velmi náročné. Tím pádem není možné detekci provést v reálném čase. Proto se tento krok provádí na základě indexovací tabulky, v níž je možné rychle vyhledávat shodné referenční body.

Indexovací tabulka o rozměrech $n_d \times n_\phi \times m$, kde m představuje počet referenčních bodů, je vytvořena v trénovací fázi. Buňka tabulky (q_d, q_ϕ, i) obsahuje seznam indexů j všech trénovacích snímků T_j, ve kterých je kvantizační hodnota vzdálenosti $d_{T_j}(p_i)$ rovna q_d a hodnota orientace $\phi_{T_j}(p_i)$ rovna q_ϕ. Z toho vyplývá, že index každého trénovacího snímků se objeví v tabulce m-krát. Vizualizace indexovací tabulky se nachází na obrázku 7.

Když máme takto sestrojenou indexovací tabulku, můžeme přejít k samotnému porovnávání jednotlivých snímků získaných průchodem skenovacího okna se všemi snímky z trénovací množiny. Abychom nalezli v tabulce indexů tu správnou trénovací předlohu, tak je nutné, aby měla určitý počet θ_v shodných referenčních bodů. Jejich počet zjistíme tak, že každý trénovací snímek sbírá hlasy za totožné referenční body. Jinak řečeno, porovnáváme kvantizační hodnoty skenovacích snímků s hodnotami v tabulce. Pro každý referenční bod vždy najdeme jednu buňku a všechny předlohy uvnitř této buňky dostanou jeden hlas. Takto postupujeme m-krát.



Obrázek 7: Vizualizace indexovací tabulky

Postup pro sestrojení indexovací tabulky je následující. Jelikož máme dva příznaky – vzdálenost a orientaci, tak budou řádky reprezentovat kvantizované hodnoty orientace a sloupce kvantizované hodnoty vzdálenosti. Jak už jsme si uvedli výše, jedná se o tabulku v trojdimenzionálním prostoru. Proto zavádíme ještě třetí rozměr tabulky, hloubku, která reprezentuje jednotlivé referenční body. Postup:

1. Definujeme počet řádků, sloupců a hloubku tabulky.

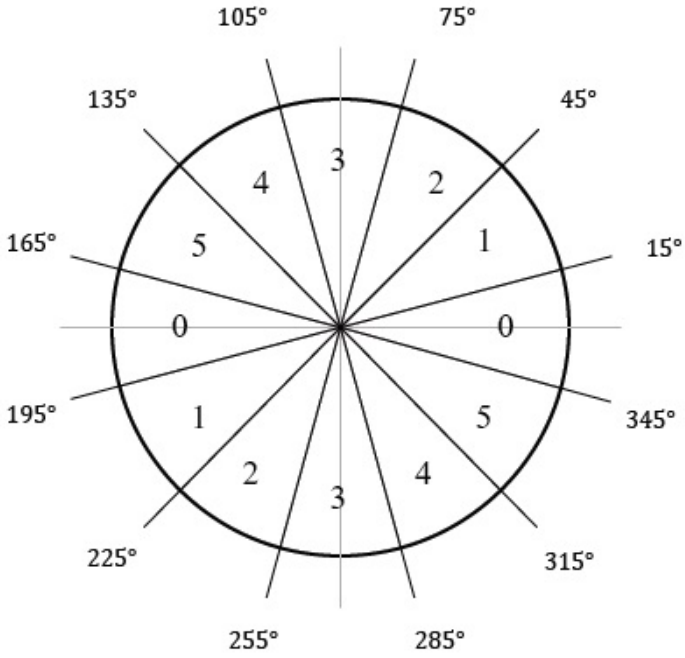
2. Přiřadíme jednotlivým buňkám intervaly hodnot, stejně jako u histogramu definujeme rozsahy jednotlivých intervalů.

3. Projdeme postupně referenční body p_i všech trénovacích obrazů. Na základě jejich kvantizačních hodnot n_d, n_ϕ a čísla referenčního bodu i, přiřadíme index obrazu do příslušené buňky. Ve výsledku bude index trénovacího snímku uložen na pozici $[x, y, z, \{seznam\_indexu\}]$. V našem případě $[n_d, n_\phi, i, \{seznam\_indexu\}]$

Při porovnávání je nutné zvolit prahovou hodnotu θ_v. Všechny snímky, které budou mít větší počet hlasů než je prahová hodnota postupují do dalšího kola, tj. do druhé fáze kaskádového

klasifikátoru. Naopak všechny snímky s nízkým počtem hlasů zahodíme a dále s nimi už nepočítáme. Z toho plyne, že v dalším kroku algoritmu pracujeme už jen s velmi malým množstvím snímků. Tím při zpracování ušetříme spoustu času i paměti a zvýšíme tak celkovou rychlost detekce.

A jak určíme rozmezí intervalů jednotlivých buněk v indexovací tabulce? V případě vzdálenosti volíme rozsah intervalů v rozmezí ideálně tří až čtyř pixelů. Počet buněk se většinou odvíjí od velikosti obrazu. Neboť čím větší obraz, tím větší distanční vzdálenost může nastat. Naopak v případě orientace víme, že úhel bude vždy nabývat hodnot od 0° do 360°. Tento rozsah lze snadno rozdělit na několik menších intervalů. Na obrázku 8 pak vidíme rozdělení intervalu pro úhly do 6 košů. Jak jste si na obrázku jistě všimli, protilehlé hodnoty orientace náleží vždy do stejných binů. Důvod je prostý. Nezáleží totiž na směru hrany, zda vede zprava doleva nebo zleva doprava, tak její orientace bude stále stejná.



Obrázek 8: Rozdělení stupnice úhlů na jednotlivé intervaly

3.8 OCM (Oriented Chamfer Matching)

Druhý stupeň ověřování je založený na použití OCM (oriented chamfer matching). Jedná se o klasický chamfer matching vylepšený o orientaci hran. Po první fázi detekování, která je založená na porovnávání histogramů a sbírání hlasů v indexovací tabulce, nám zbylo jen několik málo kandidátů na hledaný objekt. Ty nyní ověříme mnohem přesnější, ale časově náročnější metodou OCM.

Obecně bývá OCM použito pro nalezení podobných objektů v obraze. Chamfer matching funguje na principu nalezení shodných bodů ze dvou různých obrazů. Konkrétně počítáme pro každý obraz skóre, tj. počet shodných bodů. Skóre můžeme počítat pro vzdálenost s_d či orientaci s_ϕ.

$$s_d(T, I) = \frac{d_i(e) \leqslant \theta_d}{|T|} \tag{18}$$

$$s_\phi(T, I) = \frac{|\phi_T(e) - \phi_I(e)| \leqslant \theta_\phi}{|T|} \tag{19}$$

V praxi postupuje následovně. Nejprve potřebujeme z obou obrazů extrahovat hrany. Z trénovacího snímků si vezmeme jeden bod (pixel) hrany a zjistíme jeho souřadnice $[y, x]_I$. Poté se na těchto souřadnicích snažíme nalézt vzdálenost k nejbližšímu bodu (pixelu) hrany v cílovém obraze. Ke zjištění vzdálenosti využijeme distanční mapu, jejíž jednotlivé hodnoty určují vzdálenost k nejbližším pixelům hran. V klasické metodě chamfer matching sečteme vzdálenosti všech hranových pixelů. Čím nižší je tato hodnota, tím lepší je výsledek.[10] Ovšem náš algoritmus je založený na vylepšeném OCM, proto poslední krok sčítání upravíme. Když se vrátíme o krok zpět, tak známe hodnoty vzdálenosti k nejbližšímu bodu hrany. Pokud je tato vzdálenost menší než námi určená minimální hodnota θ_d, považujeme pixel hrany za totožný. Podobně postupujeme i v případě orientace. Pro pixel hrany o souřadnicích $[y, x]_I$ si zjistíme orientaci v trénovacím i testovacím obrázku. Ty od sebe navzájem odečteme a opět porovnáme s námi zvolenou minimální hodnotou θ_ϕ. Takto postupuji pro všechny hranové pixely. Nakonec výslednou sumu podělíme počtem pixelů, což označujeme jako $|T|$ a dostáváme výsledné skóre (viz rovnice (18,19)).

Poslední krok, kdy podělíme výslednou sumu počtem pixelů, který odpovídá počtu hranových pixelů v každém obraze individuálně, můžeme vylepšit. Předpokladem je, že počet hranových pixelů $|T|$ má velký rozptyl, proto může dojít ke zkreslení výsledků vzdálenosti u jednoduchých objektů, které jsou specifické nízkým počtem hranových pixelů. Abychom odstranili nechtěná zkreslení, nahradíme jmenovatele rovnicí $\lambda|T| + (1 - \lambda)\overline{|T|}$, kde$\overline{|T|} = \frac{1}{n}\sum_{i=1}^{n}|T_i|$ reprezentuje průměrný počet hranových pixelů všech trénovacích snímků a $\lambda \in [0, 1]$. Pomocí těchto vylepšení by měla detekce dosáhnout lepších výsledků než při použití klasického průměru vzdáleností.

$$s_d(T, I) = \frac{d_i(e) \leqslant \theta_d}{\lambda|T| + (1 - \lambda)\overline{|T|}} \tag{20}$$

$$s_\phi(T, I) = \frac{|\phi_T(e) - \phi_I(e)| \leqslant \theta_\phi}{\lambda|T| + (1 - \lambda)\overline{|T|}} \tag{21}$$

3.9 Výběr diskriminativních hran

Algoritmus umožňuje detekovat ve čtyřech variantách. Jednou z možných variant je výběr diskriminativních hran. Předpokladem je, že pro lepší efektivitu algoritmu a vyšší procento úspěšných detekcí nebudeme počítat OCM pro všechny hranové pixely. S každým dalším hranovým bodem totiž narůstá výpočetní složitost, což má vliv na výsledný čas. Druhým předpokladem je přesnější detekce vyhledávaných objektů. Proto ze všech dostupných bodů vybereme jen některé, pro které budeme hledat shodné body v cílovém obraze. Většina podobných algoritmů funguje na principu takovém, kdy náhodně vybere skupinu pixelů z vzorového obrazu a více se tím nezabývá. Náš algoritmus sice taky vybere skupinu pixelů, ale rozhodně ne náhodně. Hlavním kritériem při výběru pixelů je stabilita vzhledem k úhlu pohledu a orientace hran. Na základě tohoto principu vybereme jen ty nejdůležitější hranové pixely.

3.9.1 Výběr podle stability hrany

Stabilitu hran určujeme vzhledem k úhlu pohledu. Ale jelikož naše datová sada neobsahuje informace o tom, které snímky si jsou podobné, tak musíme tuto chybějící informací nahradit. A to tak, že pro každý vstupní trénovací snímek T definujeme množinu $N(T)$, což je množina nejpodobnějších snímků k původnímu T. Celý výpočet pro zjištění podobných snímků proběhne na základě předchozí rovnice OCM (20,21). Předpokladem je, že výslednou množinu $N(T)$ budou tvořit velmi podobné snímky. Na těch budou zachyceny totožné objekty, které se od sebe budou lišit jen nepatrnou změnou úhlu pohledu.

Poté co zjistíme množinu podobných snímků $N(T)$, tak postupně projdeme všechny hranové pixely každého trénovacího snímku z této množiny a jeden po druhém $e \in T$ je dosadíme do následující rovnice (22,23) a vyhodnotíme n_T. Ve výsledku zachováme jen ty pixely hran, kde $n_T(e) > \pi k$, kde $0 < \pi < 1$.

$$n_T(e) = T' \in N(T) : |d_t(e) - d_{t'}(e)| < \theta_d \tag{22}$$

$$n_T(e) = T' \in N(T) : |\phi_t(e) - \phi_{t'}(e)| < \theta_\phi \tag{23}$$

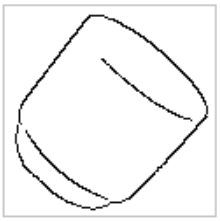
3.9.2 Výběr podle orientace hrany

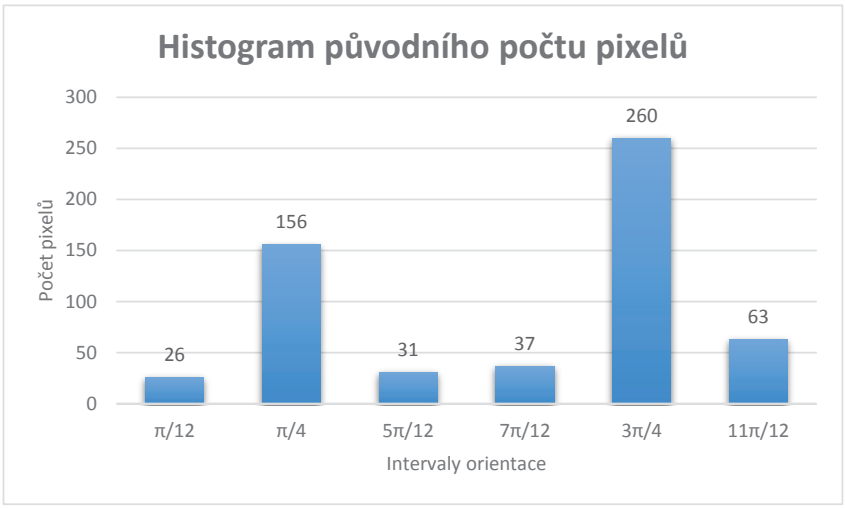
Často se setkáváme s případy, kdy v obraze převažují podélné dlouhé hrany, které obsahují velký počet pixelů se stejnou orientací. To vede ke zkreslení histogramu orientací, neboť nastane situace, kdy budou hodnoty určitých intervalů převažovat nad hodnotami jiných intervalů, což lze vidět na histogramu, který je zobrazen na obrázku 9, kde na první pohled dominují dvě přihrádky. Orientace hrany nabývá hodnot v rozmezí $0 - 360^o$. Tento rozsah rozdělímei na 12 přihrádek, To znamená, že interval každé příhrádky se pohybuje v rozmezí 30^o. Ale jelikož mají

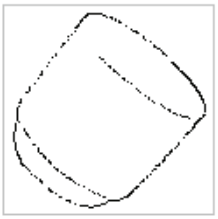
protilehlé úhly stejnou orientace hrany, sloučili jsme vždy opačné intervaly dohromady, čímž jsme se dostali na 6 přihrádek.

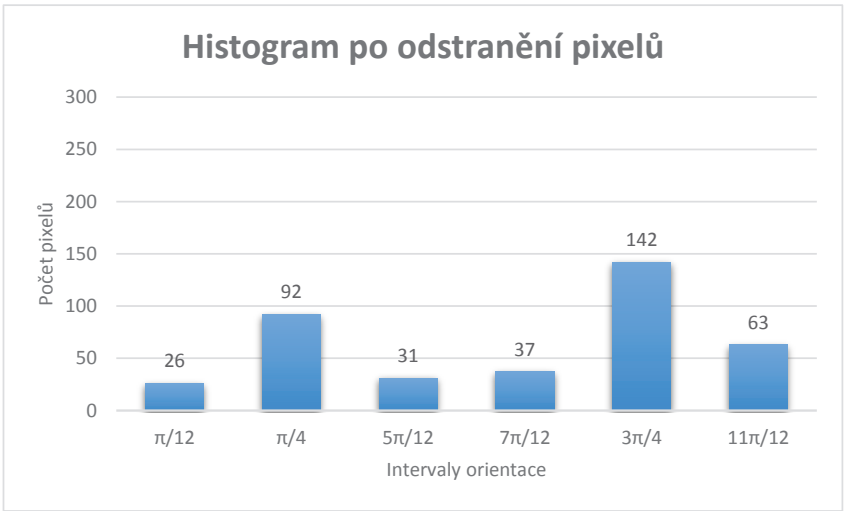
A jak vyhodnotíme danou situaci? Řešením je odstranit 40% všech hranových pixelů ve dvou nejvyšších koších. Tato metoda je podobná částečné ekvalizaci histogramu. Problém nastává v případě, kdy se rozhodujeme, které pixely z dlouhých hran odstranit. Avšak řešení v podobě náhodného generátoru se ukázalo jako dostačující.

Na obrázku níže lze vidět příklad, kdy z původního snímku s počtem 573 hranových pixelů odstraníme nadbytečné pixely ze dvou nejvyšších binů. Výsledkem je nový snímek s počtem 391 hranových pixelů. Stejným způsobem postupujeme pro všechny snímky z trénovací množiny dat.









Obrázek 9: Odstranění pixelů

4 Implementace

Cílem této práce je naimplementovat algoritmus pro detekci beztexturových objektů, který vychází z původního řešení algoritmu [11]. Teorie nezbytná pro pochopení funkčnosti algoritmu již byla vysvětlena výše. Proto se nyní zaměříme na praktický popis jednotlivých kroků, použitých metod a zvolených parametrů.

Implementace algoritmu proběhla ve vývojovém prostředí *Visual Studio 2013* s využitím grafické knihovny *OpenCV 3.1*. K naprogramování a napsání kódu celé aplikace byl zvolen objektově orientovaný jazyk *C++*.

4.1 OpenCV

OpenCV (Open Source Computer Vision) je volně šiřitelná multiplatformní knihovna pro práci s obrazem. Implementujeme ji v případě, kdy potřebujeme získat informace z obrazu a dále s nimi pracovat. V knihovně je dostupných více než 2500 optimalizovaných algoritmů, které umožňují snadnější zpracování obrazu a ulehčují programátorům práci v oblasti počítačového vidění a analýzy obrazu v reálném čase. Z toho plyne, že ji využíváme v případě, kdy potřebujeme identifikovat či segmentovat objekty od nezajímavého pozadí, sledovat body zájmu či rozpoznávat obličeje.

Knihovna je volně šiřitelná pro osobní i komerční využití. Podporuje programovací jazyky C, C++, Javu, Python a Matlab a běží na systémech Windows, Linux, MAC OS, iOS a Android. [12] Velkou výhodou knihovny je, že dokáže pracovat s více jádry a tím urychlit běh celého algoritmu, neboť právě rychlost je při zpracování obrazu klíčová. V našem případě využíváme knihovnu pro načítání a vykreslování obrazu, pro přístup k hodnotám jednotlivých pixelů a několik dalších metod pro částečné ušetření práce s obrazem.

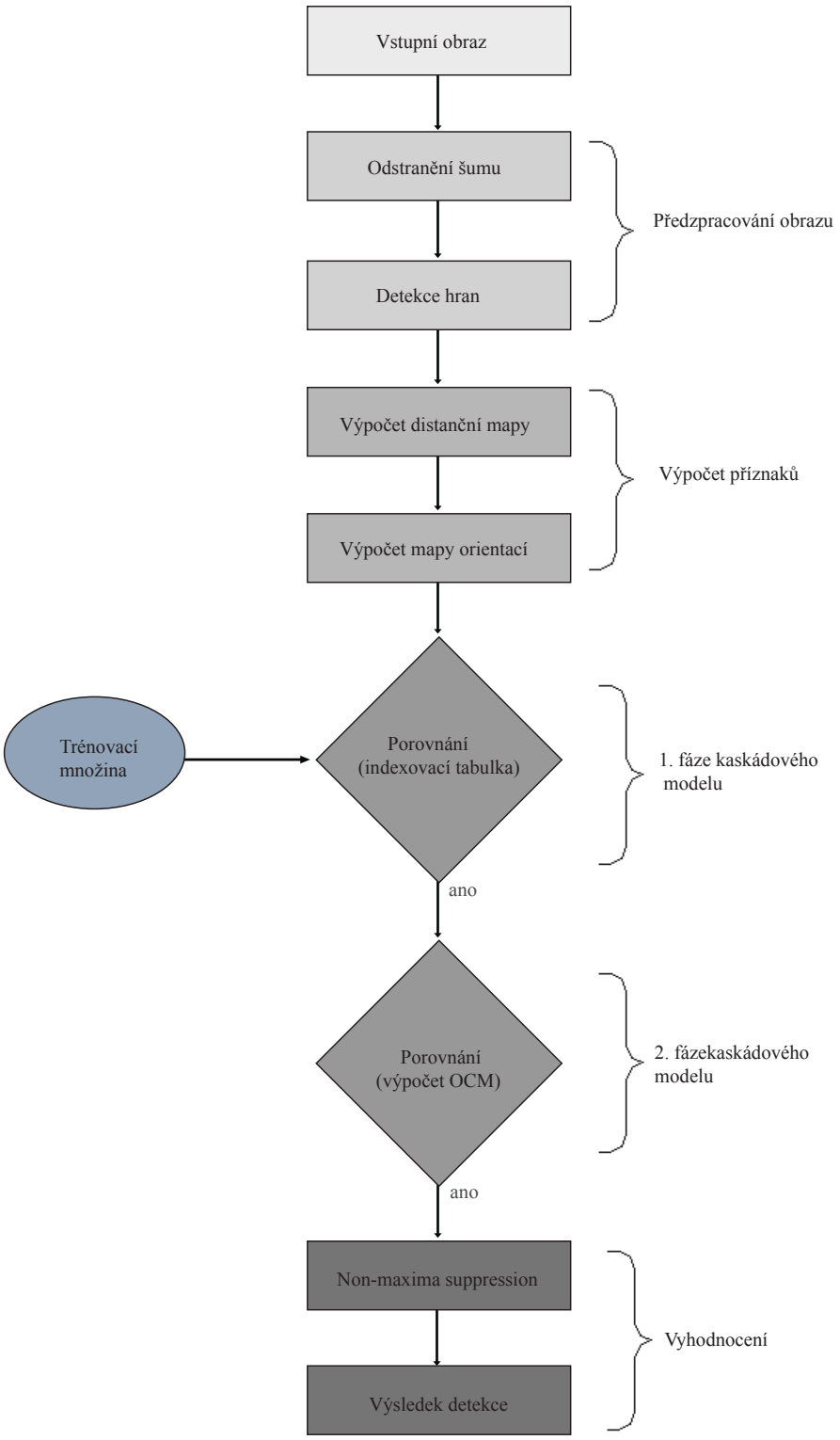
Algoritmy a funkce obsažené v knihovně původně vyvíjela v roce 1999 společnost Intel. Nyní je spravuje Willow Garage a Itseez. Tehdy bylo cílem projektu:[13]

- Šíření čitelného a přenositelného kódu.
- Komerční aplikace s volně dostupným a šiřitelným kódem.

Tyto a mnohé další požadavky byly dodrženy. Od roku 2000, kdy se poprvé algoritmy dostaly do rukou veřejnosti, vyšlo několik rozšířených a aktualizovaných verzí OpenCV. Poslední verze OpenCV 3.2 byla vydána 23. prosince 2016.

Ve svém algoritmu využívám velmi často datové typy a vestavěné metody z knihovny OpenCV. Pro ukládání informací o jednotlivých pixelech a následný přístup k nim slouží datový typ cv::Mat. Neboť většina metod z této knihovny, které jsou určené pro práci s obrazem, vyžadují pro svou funkčnost právě tento datový typ.

Diagram implementace detekčního algoritmu je znázorněn na obrázku 10.



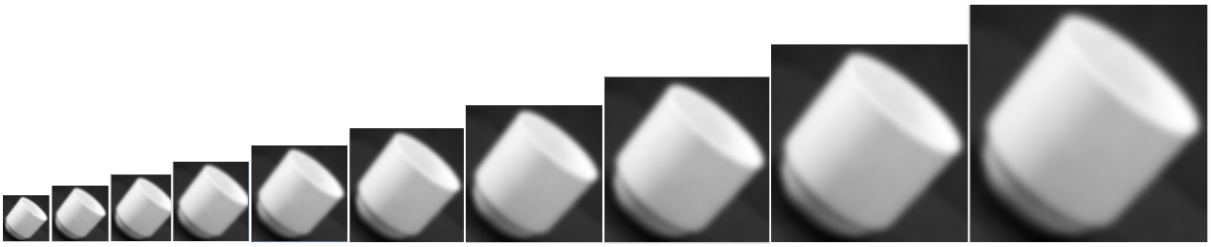
Obrázek 10: Diagram algoritmu pro detekci beztexturových objektů

4.2 Datová sada

Vstupní data tvoří obrázky, které jsou dostupné z [14]. Jednotlivé obrázky mají velikost 48x48 pixelů a zachycují objekty v odlišných pozicích a z různých úhlů pohledu. Každý objekt je umístěn na černém pozadí a postupně otáčen dokola. Poté je každý snímek rotován v rozsahu -40^o až 40^o v celkem 9 krocích. Ukázka vstupních snímků je znázorněna na obrázku 11.

Obrázek 11: Úkazky ze sady vstupních snímků

Načtení datové sady je zároveň prvním krokem algoritmu. Pro úspěšné detekování jednoho objektu je nutné načíst zhruba 1620 trénovacích snímků, na kterých je objekt zachycen v různých pozicích. Ovšem problém nastává v případě, kdy chceme detekovat objekty větší než 48×48 pixelů. Jelikož je náš algoritmus natrénovaný pouze na snímky této velikosti, tak bude náš detektor schopen rozpoznat pouze takto malé objekty. Proto každý vstupní obrázek postupně zvětšujeme až na velikost 248×248 pixelů, což je více než pětinásobná velikost původního obrazu. Postupné zvětšování je znázorněno na obrázku 12. S takto enormním zvětšením vyvstává celá řáda problémů. Jelikož obraz není pořádně zaostřený, vznikají nadbytečné hrany nebo naopak řada důležitých hran chybí. Nejefektivnějším řešením proti několika násobnému vykreslení hran se ukázalo rozmazání obrazu s využitím Gaussova filtru. Čím větší obraz vytváříme, tím větší by rozmazání mělo být. Ovšem to vše na úkor chybějících hran, které však nejsou pro samotnou detekci až tak důležité.



Obrázek 12: Postupné zvětšování vstupních snímků

V praxi využijeme vestavěnou funkci OpenCV pro změnu velikosti obrazu (viz výpis 1). Parametr označovaný jako vstupní obraz bude vždy náš nejmenší načtený snímek o velikosti 48×48 pixelů. Z toho logicky vyplývá, že výstupním snímkem bude obraz s novými rozměry. V našem případě se bude vždy jednat o obraz zvětšený. Kolikkrát se má daný snímek zvětšit nám říkají hodnoty parametrů fx a fy. Vycházíme z toho, že pro původní velikost obrazu jsou tyto hodnoty rovny číslu 1.0. Řadou experimentálních pokusů, se jako nejvhodnější hodnoty pro správný chod algoritmu jevily následující hodnoty zapsané v tabulce1.

Tabulka 1: Velikost trénovacích obrázků při postupném zvětšování

Velikost obrazu	fx	fy
Původní obraz 48×48	1.00	1.00
Obraz 58×58	1.20	1.20
Obraz 69×69	1.44	1.44
Obraz 83×83	1.73	1.73
Obraz 100×100	2.07	2.07
Obraz 119×119	2.49	2.49
Obraz 143×143	2.99	2.99
Obraz 172×172	3.58	3.58
Obraz 206×206	4.30	4.30
Obraz 248×248	5.16	5.16

```
void cv::resize(cv::Mat vstupObraz, cv::Mat vystupObraz, cv::Size(), double fx,
    double fy, int interpolace)
```

Výpis 1: Metoda pro změnu velikost obrazu

4.3 Výpočet příznaků

Nyní je náš detektor schopen rozpoznávat objekty různých velikostí. Proto můžeme přejít k trénovací fázi, ve které popíšeme každý objekt pomocí dvou příznaků - *vzdálenosti* a *orientace*. Jak už jsme si uvedli výše, příznaky počítáme jen pro konkrétní body, tj. pro body referenční mřížky. V našem algoritmu je zvolen pro referenčními body p_i pravidelný odstup čtyř pixelů. Z tabulky 2 je zřejmé, že trénovací snímky mají různou velikost. Tomu logicky odpovídá i různý počet referenčních bodů (v tabulce označován jako p). Zastoupení těchto bodů s velikostí obrazu exponencionálně narůstá. Stejně tak se zvyšuje parametr θ_v, který označuje prahovou hodnotu. Ta nám při hlasování v indexovací tabulce udává, kolik shodných referenčních bodů je potřeba, abychom mohli ke skenovacímu oknu přiřadit předlohu v podobě trénovacího snímku a tím pádem snímek postoupí do druhé fáze kaskádového modelu.

Tabulka 2: Velikost trénovacích obrázků při postupném zvětšování

Velikost obrazu	p	θ_v
Původní obraz 48×48	36	26
Obraz 58×58	49	28
Obraz 69×69	81	32
Obraz 83×83	121	40
Obraz 100×100	169	54
Obraz 119×119	256	64
Obraz 143×143	361	78
Obraz 172×172	529	105
Obraz 206×206	784	130
Obraz 248×248	1156	150

Když se vrátíme zpět k výpočtu příznaků, tak v této fázi sice už známe počet a pozice referenčních bodů $[x, y]$ v obrazech různých velikostí, ale stále neznáme distanční vzdálenost mezi těmito body a body nejbližších hran. Abychom mohli výpočet provést, potřebujeme znát *distanční mapu*, jejíž jednotlivé hodnoty reprezentují vzdálenosti k nejbližším pixelům hrany. Pro tento krok opět využijeme vestavěnou funkci knihovny OpenCV (viz výpis 2), jejímž výstupním obrazem je distanční mapa. Další její parametry udávají, zda se jedná o Manhattanskou, Euklidovskou či Čebyševovu metriku a říkají, zda se jedná o masku velikosti 3×3 nebo 5×5 pixelů. Přestože je Euklidovská metrika nejčastěji používaným typem, tak náš algoritmus pracuje s metrikou Manhattanskou, která se ukázala jako nejvhodnější pro další kroky.

```
void cv::distanceTransform(cv::Mat vstupObraz, cv::Mat vystupObraz, int
    distancniTyp, int velikostMasky)
```

Výpis 2: Funkce pro výpočet distanční mapy

Jelikož druhým příznakem je orientace, tak potřebujeme získat mapu orientací. To je obraz, jehož jednotlivé hodnoty udávají úhel orientace hrany. Toho dosáhneme využitím Sobelova filtru ve směru osy x a y a funkce, která vrátí pro každý pixel hodnotu arkus tangens y/x vyjádřenou v radiánech nebo stupních. K výpočtu tohoto kroku využijeme funkci znázorněnou ve výpisu 3 s hodnotami, které jsou zapsány v tabulce 3:

```
void cv::Sobel(Mat vstupObraz, Mat vystupObraz, int depth, int dx, int dy, int
    ksize)
```

Výpis 3: Funkce pro výpočet Sobelova filtru

Tabulka 3: Hodnoty dosazené do funkce pro použití Sobelova filtru

Obraz	dx	dy	ksize
Sobel ve směru osy x	1	0	3
Sobel ve směru osy y	0	1	3

Do funkce níže dosadíme jako parametry výstupní obrazy ve směru os x a y, které jsme získali aplikací Sobelova filtru na obrázek ve stupních šedi. Výsledkem je úhel vyjádřeny v radiánech.

```
float atan2(float y, float x)
```

Výpis 4: Funkce pro výpočet arkus tangens

Nyní jsme sice ve fázi, kdy známe na základě distanční mapy hodnotu vzdálenosti nejbližších hranových pixelů, ale neznám jejich souřadnice, tudíž k nim nemůžeme přiřadit správnou velikost orientace hrany. K tomu bylo třeba vymyslet a naimplementovat metodu, která tento problém vyřeší a zjistí souřadnice $[x, y]$. Prvotním nápadem byl rekurzivní algoritmus, který si v každém kroku zjistí hodnotu čtyř okolních sousedů, vybere hodnotu nejmenšího z nich a přesune se na jeho pozici. Odtud pokračuje rekurzivně stejným principem dále, dokud nenarazí na souseda s hodnotou 0. To znamená, že narazil na bod hrany. Tento rekurzivní princip byl sice správný, ovšem poměrně zdlouhavý, což se projevovalo na výsledném čase algoritmu. Proto bylo nutné vylepšit tento krok. Princip nové metody je popsán v algoritmu1.

Algorithm 1 Spočítej souřadnice $[x, y]$ nejbližšího hranového pixelu

1. Spočítej distanční mapu.
2. Pro každý bod referenční mřížky p_i opakuj:
3. Zjisti hodnotu distanční vzdálenosti v bodě p_i.
4. Podívej se do všech čtyř směrů.
5. Zjisti, který z pixelů posunutý o danou distanční vzdálenost je roven číslu 0.
6. Vrať souřanice $[x, y]$ nulového pixelu.

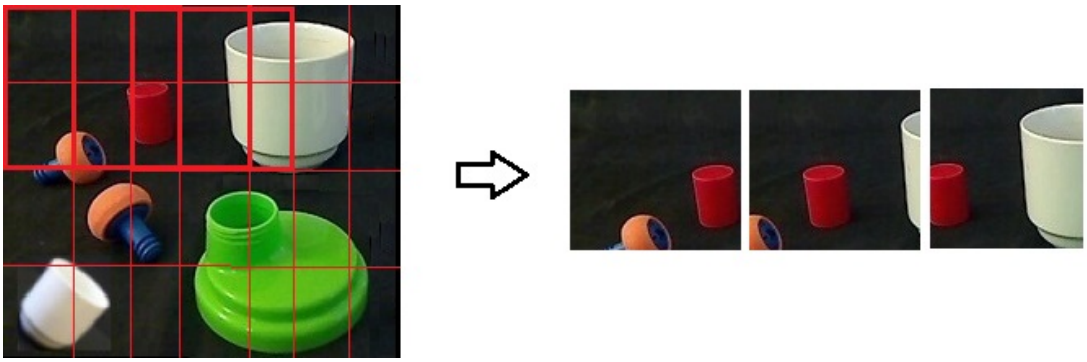
Když už souřadnice nejbližšího hranového pixelu známe, stačí zjistit hodnotu, která se skrývá na těchto souřadnicích v mapě orientací. Teď už známe hodnoty příznaků vzdálenosti a orientace pro každý bod referenční mřížky. Pro snadnější práci a lepší přehlednost si uložíme jednotlivé hodnoty do vektoru příznaků.

4.4 Posuvné okno (sliding window)

Když máme natrénováno, přichází na řadu práce s cílovým obrázkem. Ten vždy obsahuje několik navzájem různých objektů. Našim úkolem je detekovat a rozpoznat objekty podobné těm, které jsme v předchozí fázi natrénovali. Mohli bychom je snadno a rychle porovnat na základě podobnosti histogramů, ovšem problém nastává v různých velikostech. Jelikož testovací snímky mají v průměru rozlišení 640×480 pixelů, musíme obraz rozdělit na menší obrázky. Ideálně tak,

aby jejich velikost odpovídala rozměrům trénovacích snímků. K tomu využijeme postup, který je označován jako skenovací či posuvné okno.

Skenovací okno je obdélníková či čtvercová oblast, s pevně danou šířkou a výškou, která postupně projíždí obraz zleva doprava a zároveň shora dolů, jak je vidět na obrázku 13. V našem případě projdeme obraz skenovacím oknem hned několikrát, neboť i trénovací snímky máme v různých velikostech a po každém průchodu je potřeba posuvné okno zvětšit.



Obrázek 13: Skenovací okno

Pro každý výřez skenovacího okna postupujeme stejně jako v případě trénovacích snímků. Opět vypočítáme příznaky vzdálenosti a orientace a uložíme je do vektoru příznaků, kde čekají na další zpracování.

4.5 Indexovací tabulka

Abychom nezatěžovali paměť zbytečnými obrázky, které vznikly jako výřez skenovacího okna a jejich vektory příznaků, tak provedeme ihned porovnání každéhu výřezu s trénovacími snímky. K tomu využijeme již dobře známý postup, založený na sbírání hlasů v indexovací tabulce. V teoretické části jsme si vysvětlili co to indexovací tabulka je, jak se tvoří a jak jednotlivé buňky sbírají hlasy. V praxi bylo nutné při programování algoritmu vymyslet strukturu, která ze známých hodnot vzdálenosti a orientace vytvoří trojdimenzionální tabulku, která uvnitř buňky ukládá seznam trénovacích snímků.

Jako nejjednodušší se jevil postup, při kterém byla sestavena trojdimenzionální tabulka o velikosti $[výška][šířka][hloubka]$, kde se ke každé buňce přistupuje skrz souřadnice $[x][y][z]$. Tato úvaha byla sice správná a funkční, ovšem pro větší skupinu dat ne příliš efektivní. Jako nejideálnější řešení se ukázalo převést trojdimenzionální tabulku na jednodimenzionální. Velikost této tabulky určíme vynásobením hodnot výšky, šířky a hloubky. Jak vypadá samotná deklarace je možné spatřit ve výpise 5. Pro přístup k jednotlivým buňkám je použitý vzorec, ve kterém se násobí jednotlivé souřadnice s původní výškou a šířkou tabulky. Postup násobení je znázorněn ve výpisu 6.

```
vector<int> *pole = new vector<int>[vyska * sirka * hloubka];
```

Výpis 5: Deklarace jednodimenzionálního pole

```
int<vector> prvkyBunky = pole[x + vyska* (y + sirka* z)]
```

Výpis 6: Přístup k jednotlivým buňkám pole

4.6 Chamfer matching

V této druhé části našeho algoritmu se snažíme o výběr nejvíce podobných kandidátů s vzorovými snímky. Součástí výpočtu je určení procentuální shody těchto dvou snímků. Jak jsme si uvedli v teoretické části, k tomuto kroku využijeme metodu zvanou chamfer matching. Ta je sice časově náročná, ale jelikož ji použijeme jen na malý počet snímků, tak výsledný čas algoritmu přílis nezatíží.

Prvním krokem této metody je zjištění pozice $[x, y]$ všech hranových pixelů. K tomu využijeme opět jednu z funkcí OpenCV, kterou vidíme ve výpise 7. Vstupní obraz této funkce je v binárním tvaru, kde je pixel hrany označen hodnotou 0 a pixel pozadí hodnotou 1. Výstupem je matice, jejíž hodnoty tvoří souřadnice $[x, y]$ všech nulových pixelů, tj. pixelů hran.

Cyklem postupně projdeme jednotlivé souřadnice a vždy se podíváme, jaká hodnota se nachází na této pozici v trénovacím a testovacím obraze. Tyto čísla dosadíme do rovnice pro chamfer matching (viz vzorce (20,21)), kde je navzájem odečteme a výslednou hodnotu porovnáme s námi zvolenou hranicí shody. Pokud je číslo větší než naše hranici, pixely výřezu i trénovacího snímku považuju za totožné a navýším hodnotu celkového skóre.

```
void cv::findNonZero(cv::Mat vstupniObraz, cv::Mat vystupniObraz)
```

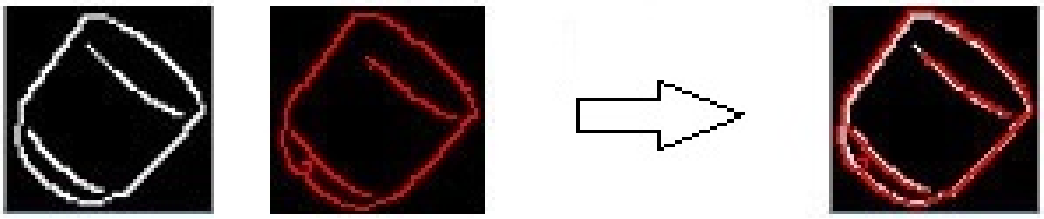
Výpis 7: Zjištění pozice pixelů hrany

Tabulka 4: Hodnoty skóre pro chamfer matching

	Hranice shody
Skóre s_d	4.0
Skóre s_ϕ	30^o

Výsledkem rovnice je skóre, tj. počet shodných bodů pro vzdálenost s_d a skóre pro orientaci s_ϕ. V tomto kroku už stačí jen vybrat snímek, který má nejvyšší skóre v obou kategoriích a tento výřez vyznačit červeným rámem na původním cílovém snímku. Součástí výstupu je i procentuální shoda výřezu a vstupního snímku. Tímto krokem, kdy dojde k vyznačení detekovaných objektů končí druhý stupeň kaskády a zároveň celý výpočet algoritmu.

Obrázek 14 zachycuje reálné výsledky chamfer matchingu. Zleva vidíme vstupní trénovací obrázek v hranovém tvaru a vedle něj výřez z testovacího obrázku, který není na první pohled úplný a pravidelný, neboť je poškozen šumem. Vpravo pak vidíme, jak by vypadal obraz, kdybychom oba předchozí navzájem překryli.



Obrázek 14: Chamfer matching

4.7 Výběr diskriminativních hran

Ovšem v mnoha případech je hranice objektů tvořena velkým množstvím pixelů. Vznikají nadbytečné hrany v místě šumu, které sice lidské oko jako hrany nevyhodnotí, ale počítač ano. Proto musíme jeho chyby napravit. Odstraníme tedy nadbytečné hrany metodou, jejíž princip funguje na základě stability hrany. Na takto upravený obrázek použijeme další metodu, která odstraní pixely z dlouhých rovných hran, které by mohly zkreslovat histogram. Jak vypadá postup pro odstanění pixelů podle orientace hrany vidíme v následujícím algoritmu 2.

Algorithm 2 Výběr pixelů podle stability hrany

1. *Vstup:* cannyho snímek a původní obrázek.
2. Zjistíme souřadnice $[x, y]$ hranových pixelů a jejich celkový počet.
3. Vypočteme orientaci hran a z hodnot vytvoříme histogram.
4. Vzájemným porovnáním nalezneme dva nejvyšší koše v histogramu.
5. Z celkového počtu hodnot v nejvyšších binech odečtu 40% pixelů a odstraním je tak, že:
6. Vytvoříme si generátor náhodných čísel z intervalu $< 1, 10 >$.
7. Ke každému pixelu z nejvyšších binů vygeneruju náhodné číslo.
8. Jestliže je číslo menší než hodnota 4, tak hranový pixel odstraním.

Princip generátoru náhodných čísel je založen na tom, že v případě, kdy generujeme čísla z intervalu 1 - 10, tak máme právě 40% šanci, že padne číslo čtyři a nižší. K implementaci generátoru náhodných čísel využijeme opět vestavěné funkce knihovny OpenCV. V prvním kroku deklarujeme generátor, jak lze vidět ve výpisu 8. V druhém kroku nastavíme generátor tak, aby vypisoval čísla z intervalu 1 až 10 (viz výpis 9).V třetím kroku je znázorněno už samotné generování náhodných čísel, které nám vrátí konkrétní hodnotu (viz výpis 10). Pokud je hodnota vyšší než číslo čtyři, tak bod zůstává pixelem hrany. V opačném případě dojde k jeho odstranění. Výsledné obrázky po odstranění dvou nejvyšších binů jsou znázorněny na obrázku 15.

```
std::default_random_engine generator;
```

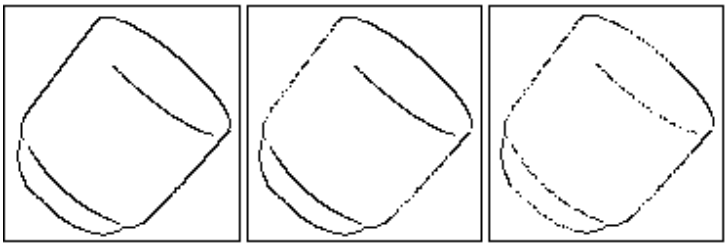
Výpis 8: Deklarace generátoru náhodných čísel

```
uniform_real_distribution<double> distributionreal(1, 10);
```

Výpis 9: Nastavení generátoru náhodných čísel

```
int hodnota = distributionreal(generator);
```

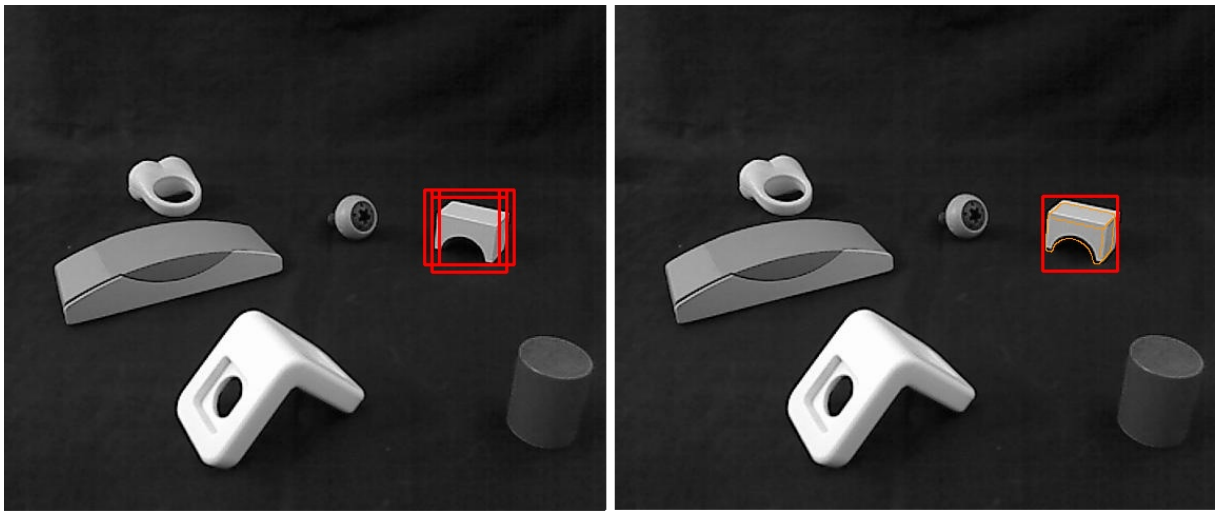
Výpis 10: Vygenerování náhodné hodnoty



Obrázek 15: Původní obrázek (vlevo), po odstranění nejvyššího binu (uprostřed), po odstranění dvou nejvyšších binů (vpravo)

4.8 Non-maxima suppression

Jelikož používáme pro detekci objektů v obraze posuvné okno, tak je více než pravděpodobné, že nám tato metoda vrátí skupinu detekčních oken se vzájemnými přesahy. K řešení tohoto problému a výběru co nejpřesnějšího detekčního okna bez přesahu se využívá princip zvaný non-maxima suppression. Cílem metody je tedy lokalizovat pozici objektu a potlačit ty, které jsou méně přesné.



Obrázek 16: Non-maxima suppression

Jak lze vidět na obrázku 16, metoda non-maxima suppression vybrala z řady kandidátů detekčních oken ten nejpřesnější. Výběr probíhá na základě *skóre*, které udává hodnotu podobnosti skenovacího okna s trénovacím snímkem získanou na základě výpočtu chamfer matching (viz rovnice (22,23)). Součástí detekce je lokalizace a určení pozice objektu, což lze vidět na červeně vykresleném rámu a oranžově zbarvených hranách, které kopírují hrany vzorového snímku, s nímž našel algoritmus největší shodu. Postup pro vyhodnocení metody non-maxima suppression je znázorněn v algoritmu 3.

Algorithm 3 Non-maxima suppression

1. Zjistit pozice všech detekčních oken (angl. bounding box).
2. Vytvořit pole, do kterého uložíme pozice detekčních oken seřazené podle velikosti skóre.
3. Projdi pole detekčních oken:
6. Vezmeme první bounding box a začneme jej porovnávat s ostatními.
7. Jestliže se bounding boxy překrývají, druhý z nich zahodíme.
8. Jestliže se bounding boxy nepřekrývají anebo jen z části, ponecháme je oba.

Jak je v algoritmu uvedeno, je nutné nejprve setřídit množinu detekčních oken na základě velikosti jejich skóre. Tj. na první pozici bude detekční okno s nejvyšším skóre a naopak posledním prvkem bude detekční okno s nejnižším skóre. Když máme takto setříděné pole, přejdeme k samotnému porovnávání, které je založeno na *průniku* dvou čtverců. Vezmeme tedy bounding box, který se nachází vlevo a provedeme výpočet průniku se všemi bounding boxy na pravo od něj. Pokud je plocha jejich průniků větší než 75%, tak druhý, ten méně podobný bounding box, zahodíme. Naopak pokud je plocha jejich průniku zanedbatelná, vykreslíme červeným rámem i druhé detekční okno s menším skóre a zároveň postupujeme rekurzivně dále. To znamená, že pro tento právě vykreslený prvek opět spočítáme plochu průniků se všemi prvky napravo od něj. Tím docílím toho, že nám v obraze zůstanou červeně vyznačená pouze ta detekční okna, která se nepřekrývají.

Otázkou zůstává, jak vypočteme obsah plochy průniku dvou čtverců (jak lze spatřit na obrázku 17) v programovacím jazyce C++? Jednoduše. Známe souřadnice x, y čtvercového detekčního okna. To nám umožňuje využít obdélníkovou strukturu z knihovny OpenCV (viz výpis 11). Nad touto strukturou pak můžeme používat *bitové operátory*. Konkrétně operátory sloučení a průniku. Sloučení reprezentuje logické AND a značíme jej operátorem |. Průnik reprezentuje logické OR a značíme jej operátorem &. Nyní si představme, že máme obdelníky A a B. Ty reprezentují dvě detekční okna, která se navzájem překrývají. Výsledným průnikem je obdélník C. Implementace jak docílíme průniku použitím bitových operací je znázorněna ve výpisu 12. Nyní nám stačí už jen zjistit velikost plochy obdélníku C. K tomu opět využijeme jednu z vestavěných funkcí knihovny OpenCV (viz výpis 13). V konečné fázi porovnáme velikost plochy obdélníku C s plochou obdélníku A. Pokud je první zmíněná hodnota ve srovnání s druhou zmíněnou hodnotou podstatně menší, tak jednoduše ponecháme obě detekční okna A i B. V opačném případě ponecháme pouze detekční okno A.

```
cv::Rect(int x, int y, int sirka, int vyska);
```

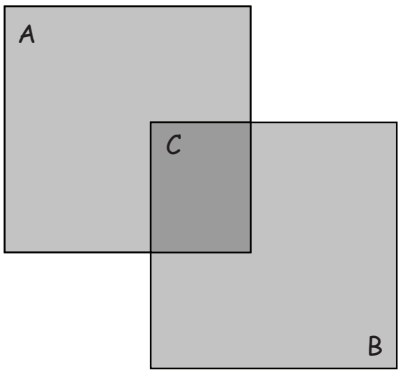
Výpis 11: Funkce pro vytvoření obdélníku

```
cv::Rect C = A | B;
cv::Rect C = A & B;
```

Výpis 12: Implementace bitových operací

```
int obsahPlochy = C.area();
```

Výpis 13: Funkce pro výpočet velikosti plochy obdélníka

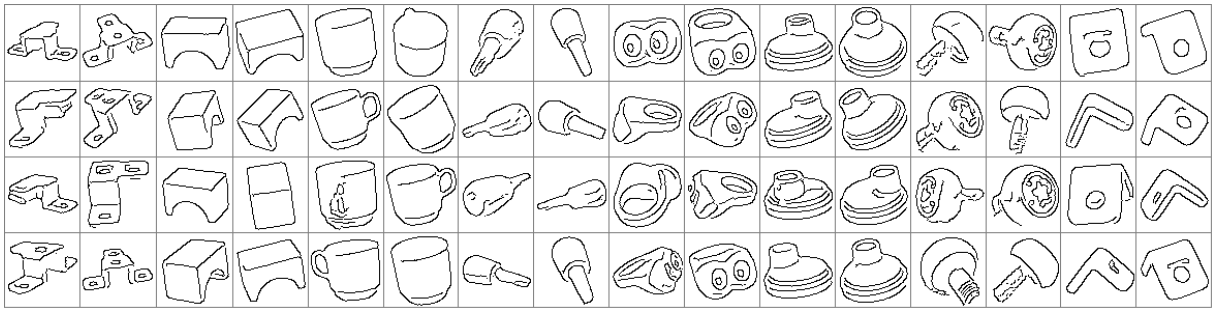


Obrázek 17: Znázornění průniku dvou detekčních oken A a B

5 Dosažené výsledky

Jelikož je náš algoritmus založen na detekci hran, tak je nutné ve všech načtených obrázcích v prvním kroce rozpoznat hrany. To platí jak pro trénovací, tak i pro testovací snímky. Jak už jsme si uvedli výše, datová sada obsahuje celkem osm objektů - *blok, most, hrnek, šroubovák, oči, víko, šroub a bílý blok*. Každý z těchto objektů vyskytuje v trénovací datové sadě přesně 1620x. Na obrázku 18 lze vidět ukázku detekce hran všech osmi objektů z trénovací sady.

Cílem datové sady je, aby obsahovala navzájem různé objekty. Proto je už na první pohled možné spatřit, že objekty se od sebe liší tvarem, zaoblením a složitostí. Například blok vlevo je příliš komplikovaný. Nejrůznější výčnělky, díry a tvar predikují velmi obtížné rozpoznání. Naopak most, hrnek či bílý blok mají natolik jednoduché tvary a primitivní hrany, že detekce těchto objektů by neměla činit problémy. Ovšem takto obecné a jednoduché hrany by mohly mít problém v případě mnoha falšených detekcí. Kompromisem mezi jednoduchostí a složitostí, specifickými a obecnými hranami je šroubovák, oči a víko. V případě detekce očekáváme u těchto objektů vyšší procento úspěšnosti.

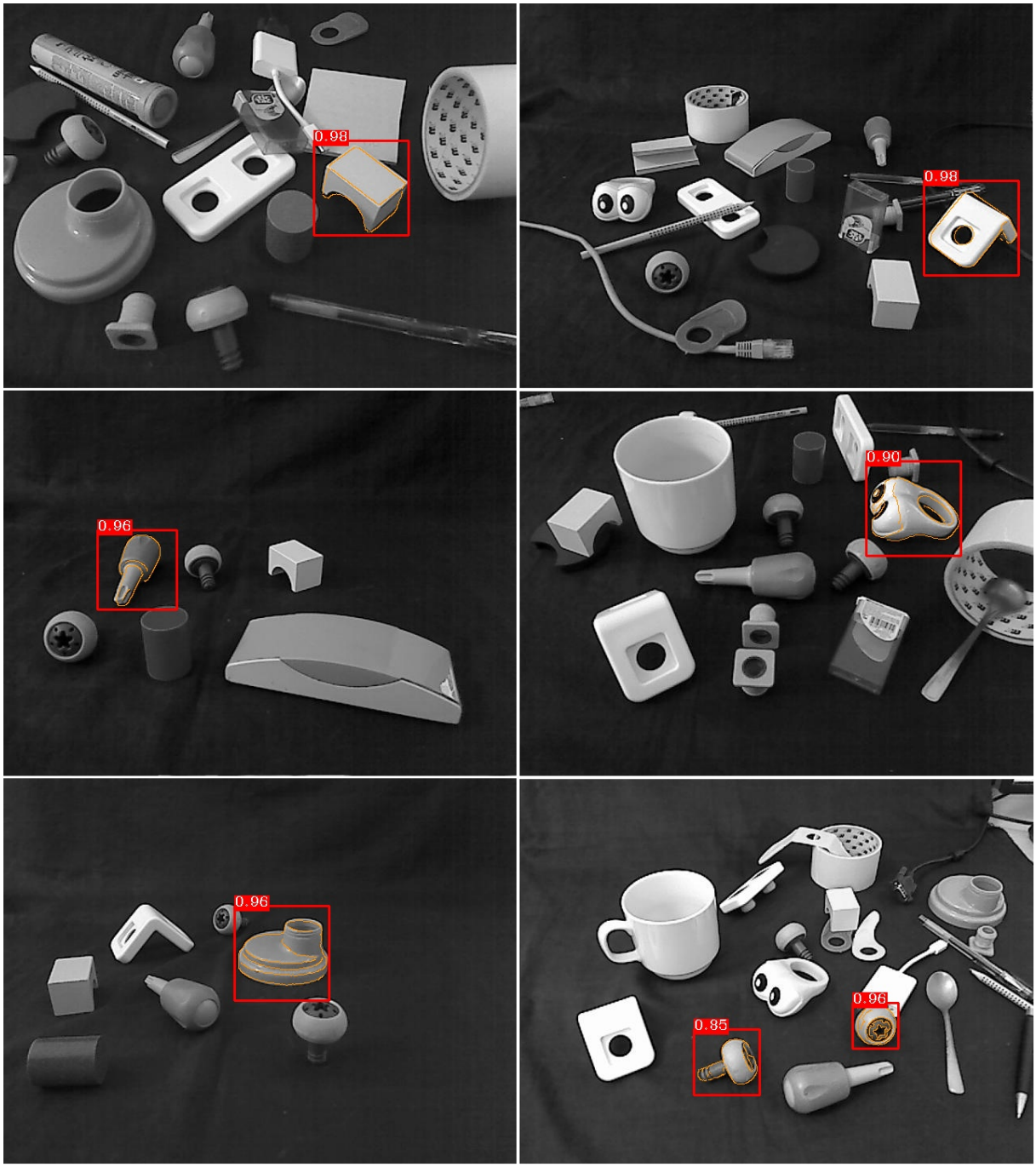


Obrázek 18: Ukázka trénovacích snímků z datové sady *CMP-8objs*

Všechny trénovací snímky se nachází na černém pozadí, proto nečiní detekce hran žádné potíže. Problém nastává v případě testovacích snímků, kde 30 z nich má pozadí černé a 30 bílé. Navíc v rámci těchto dvou skupin mají snímky různou hodnotu úrovně jasu, což způsobuje, že jsou objekty ve scéně různě nasvícené. Proto je potřeba při detekci hran zvolit pro každou skupinu snímků různé hodnoty prahu. Jelikož celá detekce závisí právě na rozpoznání hran, je tento krok jednou z nejdůležitějších částí celého algoritmu. Právě na této části by se dalo do budoucna pracovat více do hloubky a udělat detektor invariantní vůči jasu a osvětlení, protože se správnou detekcí hran roste procento úspěšné detekce.

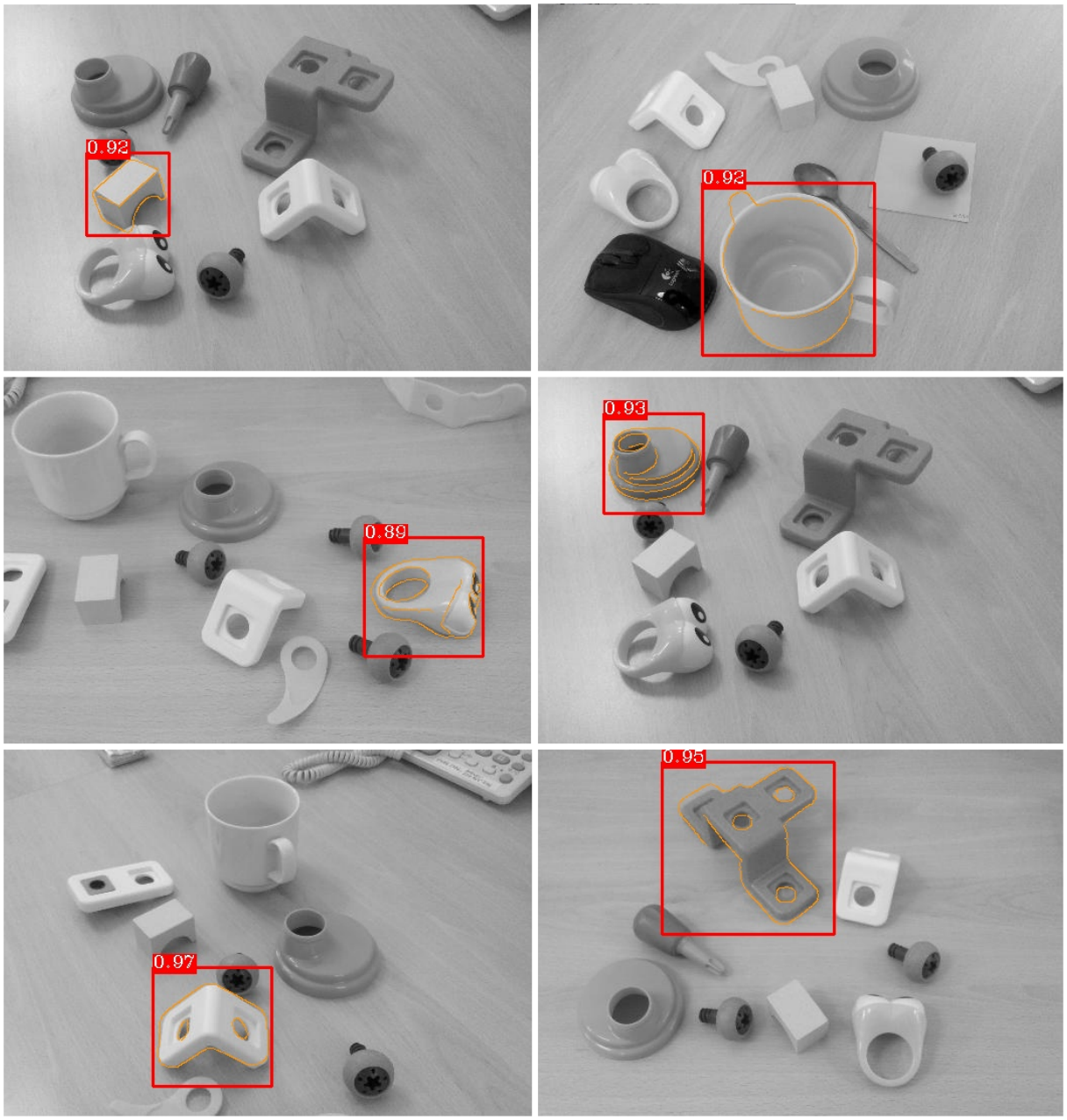
Navíc není algoritmus nativně invariantní vůči rotaci. To znamená, že pokud budeme chtít testovat algoritmus na jakékoliv jiné datové sadě, musíme na vstup dodat trénovací snímky, které budou obsahovat natočené objekty ze všech stran. Což se může jevit jako nevýhoda, ale skutečnou výhodou je potřeba jednoho trénovacího snímku na jednu pozici objektu. Což u detekčních algoritmů podobného typu nebývá zvykem. Většinou jsou zapotřebí desítky trénovacích snímků, na kterých bývá objekt zachycen ze stejného úhlu pohledu.

Obrovskou výhodou oproti podobným algoritmům je skutečnost, že typ nalezeného objektu závisí na vstupních datech trénovací množiny. To umožňuje rozšířit aplikaci algoritmu do mnoha odvětví, neboť náš detektor bude vždy vyhledávat přesně ty objekty, které jej naučíme. Na obrázku 19 je ukázka výsledné detekce. Na jednotlivých snímcích vidíme vyznačené objekty včetně určení výsledné pozice a číselnou hodnotu, která udává podobnost k nalezenému vzoru.



Obrázek 19: Výsledná detekce objektů včetně určení pozice na tmavém pozadí

Snímky na obrázku 20 zachycují objekty na světlém pozadí. A právě zde nastává problém v případě detekce hran, neboť objekty už nejsou s pozadím v kontrastu, ale doslova s ním splývají. Zkrátka hrany objektů na tmavém pozadí více vyniknou a na bílém zaniknou. Řešením je již ve fázi předzpracování obrazu vypočítat celkovou hodnotu úrovně jasu v obraze. Pokud je hodnota jasu vysoká, tak jsou pozadí i objekty vykresleny ve světlých tónech. Proto na takový obraz aplikujeme metody, které budou využívat jiných prahových hodnot než v prvním případě.



Obrázek 20: Výsledná detekce objektů včetně určení pozice na světlém pozadí

5.1 Vyhodnocení

Vyhodnocení úspěšnosti našeho detektoru vychází z matice záměn. Ta je znázorněná v tabulce 5. Podstatou této matice je vyhodnotit počet správně a špatně detekovaných objektů. Z toho vyplývá, že se jedná o binární klasifikaci. Buď je objekt detekován správně a označujeme jej *true* nebo je detekován chybně a označujeme jej jako *false*. Na základě těchto poznatků jsme schopni sestrojit a následně vyhodnotit matici záměn. [15]

Počet objektů: x	Předpověď: pozitivní	Předpověď: negativní
Skutečnost: pozitivní	TP	FN
Skutečnost: negativní	FP	TN

Tabulka 5: Matice záměn

Celkem rozlišujeme čtyři případy, které mohou u matice záměn nastat:

- *TP (true positive)* - objekty, které algoritmus detekuje správně
- *FP (false positive)* - objekty, které algoritmus detekuje chybně
- *FN (false negative)* - objekty, které se ve scéně vyskytují, ale algoritmus je nedetekuje
- *TN (true negative)* - objekty, které se ve scéně nevyskytují a algoritmus je nedetekuje

Na základě těchto vlastností jsme schopni vyjádřit celou řadu ukazatelů, kteří zjišťují výsledky na základě hodnot precision, recall, miss rate a F-score v čtyřech variantách algoritmu.

- *Precision* nebo-li přesnost vyjadřuje poměr všech rozpoznaných objektů ku všem správně detekovaným objektům. Jinak řečeno, jedná se o pravděpodobnost, která říká, jaká je šance, že se v detekované oblasti skutečně nachází hledaný objekt. Výpočet je znázorněn v rovnici 24.

$$\text{precision} = \frac{TP}{TP + FP} \tag{24}$$

- *Recall* nebo-li úplnost vyjadřuje pravděpodobnou úspěšnost, která říká, na kolik je detektor schopen zachytit námi vyhledávaný objekt. Výpočet vychází z rovnice 25.

$$\text{recall} = \frac{TP}{TP + FN} \tag{25}$$

- *Miss rate* je protikladem recall. Vyjadřuje tedy pravděpodobnou neúspěšnost. Tj. na kolik není náš detektor schopen zachytit rozpoznávaný objekt (viz rovnice 26).

$$\text{miss rate} = \frac{FN}{TP + FN} \tag{26}$$

- *F-skóre* udává celkovou úspěšnost dektekce, která vyjadřuje přesnost detekce. Vypočteme jej podle vzorce 27 jako harmonický průměr na základě hodnot precision a recall.

$$\text{F-score} = 2 * \frac{precision * recall}{precision + recall} \tag{27}$$

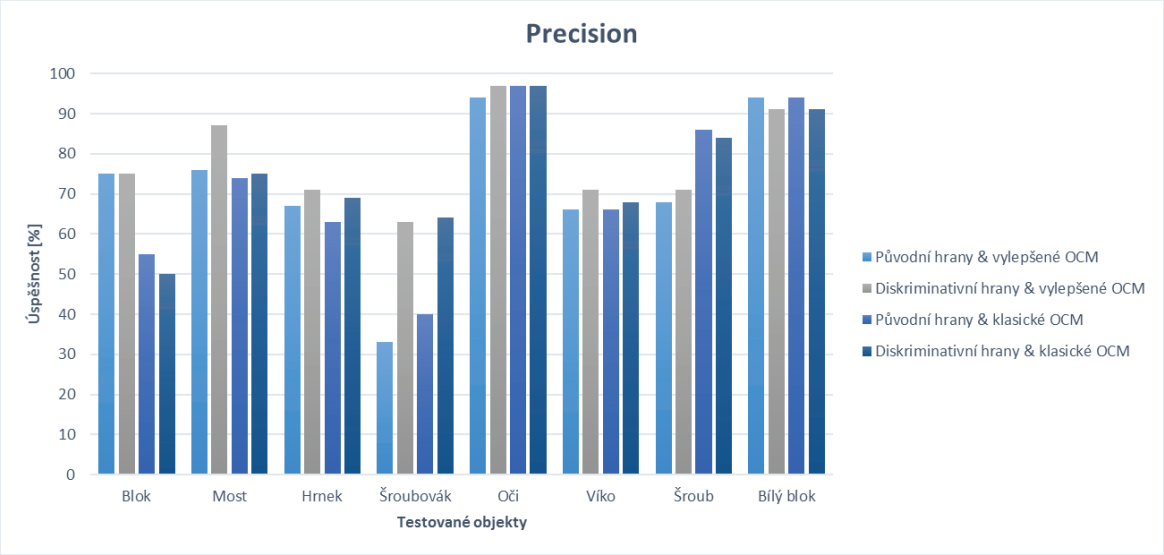
5.2 Testování

Testování proběhlo na snímcích dostupných na adrese [16]. Detekční algoritmus lze spustit a testovat ve čtyřech variantách, kde máme na výběr z následujících možností:

1. Originální hrany a výpočet vylepšeného OCM.
2. Výběr diskriminativnách hran a výpočet vylepšeného OCM.
3. Originální hrany a a výpočet klasického OCM.
4. Výběr diskriminativních hran a výpočet klasického OCM.

Aplikovat můžeme, ale nemusíme vylepšení OCM (oriented chamfer matching) a stejně tak výběr diskriminativních hran. Ovšem předpokladem je, že právě na základě těchto vylepšení bude detekce předmětů v obrazech přesnější. Jak lze vidět v tabulkách 6, 7, 8, tato hypotéza je pravdivá. Obě tyto techniky přispěly ve většině případů k větší úspěšnosti detekce.

Obecně jsou kladeny na algoritmy pro detekci a analýzu obrazu stále větší nároky. Nejdůležitějším z nich je bezesporu přesnost detekce. Ovšem pro spoustu algoritmů je podstatná také rychlost, neboť potřebujeme zpracovávat a vyhodnocovat údaje v reálném čase. Proto jsem výsledný kód prošla nástrojem pro ladění a optimalizaci algoritmů *CodeXL*. Z toho vyplynulo, že největší zpomalení algoritmu je v druhé fázi kaskádového modelu. Tedy v místě, kde dochází k výpočtu OCM. Proto je nutné, abychom se v první fází kaskádové modelu, kde je použito hlasování pomocí indexovací tabulky, zbavili co největšího množství zbytečných snímek. Čím více jich vyřadíme, tím bude výsledný čas algoritmu kratší. Ovšem indexovací tabulka není jediná metoda, kterou lze pro zbavení nepodstatných skenovacích oken použít. Oblíbeným řešením je využití k-D stromu, kde je tvar objektu reprezentován jako seskupení hran tak, aby šlo rychle a účinně ve stromu vyhledávat. Často se lze setkat také s řešením, které využívá hashovací tabulku, hierarchické vyhledávání či různé paralelní techniky. Oproti tomu náš algoritmus využívá rychlé vyhledávání v indexovací tabulce.



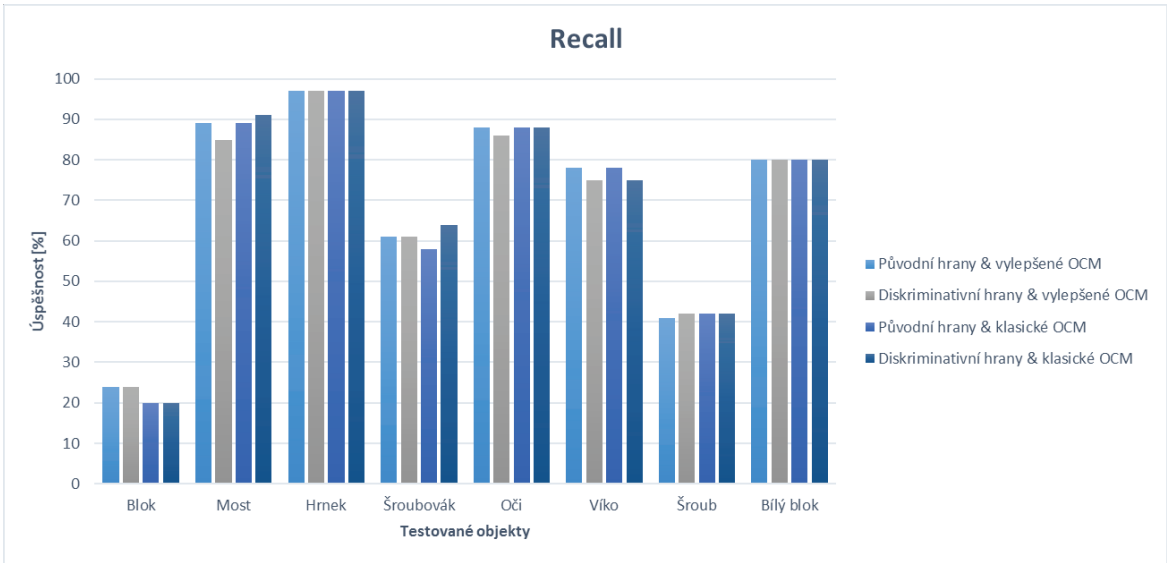
Obrázek 21: Přesnost detekce

Tabulka 6: Přesnost (precision) jednotlivých variant

Varianta	Precision
Originální hrany a výpočet vylepšeného OCM	71.63%
Výběr diskriminativnách hran a výpočet vylepšeného OCM	78.25%
Originální hrany a a výpočet klasického OCM	71.88%
Výběr diskriminativních hran a výpočet klasického OCM	74.75%

Precision nebo-li přesnost vyjadřuje počet správně detekovaných objektů ku všem detekovaným objektům. Z grafu 22 vyplývá, že nejvyšší přesnosti bylo dosaženo u detekce očí a bílého bloku. Ty jsou svým tvarem a zakřivením natolik specifické, že je téměř nemožné je zaměnit s jinými objekty. Ale zároveň jsou jejich tvar a výsledné hrany natolik jednoduché, aby je bylo možné bez potíží detekovat.

Původní hypotézou bylo, že na základě diskriminativních hran a vylepšeného OCM bude detekce přesnější. Jak lze vidět v tabulce 6, tak hypotéza je skutečně pravdivá. V případě, kdy jsme pro detekci využili obou vylepšení, dosáhla detekce úspěšnosti 78,25%. Oproti tomu algoritmus bez použití jakýchkoliv vylepšení dosáhl úspěšnosti pouze 71,88%. Takový rozdíl už není zanedbatelný.



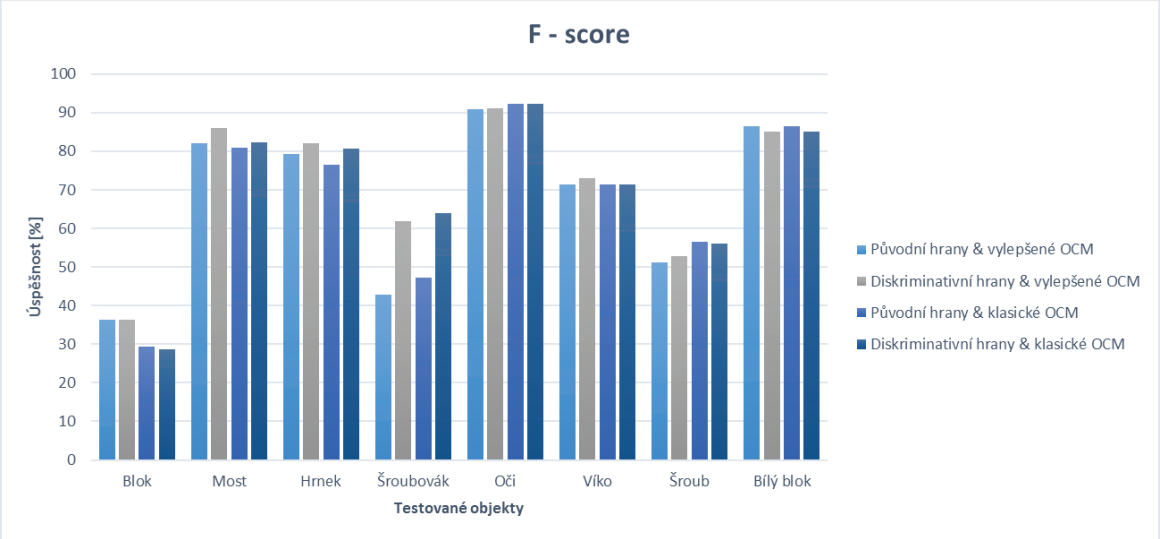
Obrázek 22: Úplnost detekce

Tabulka 7: Úplnost (recall) jednotlivých variant

Varianta	Recall
Originální hrany a výpočet vylepšeného OCM	69.75%
Výběr diskriminativnách hran a výpočet vylepšeného OCM	68.75%
Originální hrany a a výpočet klasického OCM	69.00%
Výběr diskriminativních hran a výpočet klasického OCM	69.63%

V kolika případech detektor objekt zachytil a v kolika případech jej minul zachycuje funkce recall. Z výsledků tabulky 7 i hodnot grafu 22 je na první pohled patrné, že až tolik nezáleží na tom, kterou ze čtyř variant pro detekci použijeme. Z toho vyplývá, že pokud je nalezení objektů složité, tak mu ani žádná vylepšení příliš nepomohou.

Pokud se vyhledávaný objekt ve scéně nachází, tak je šance téměř 70%, že jej algoritmus skutečně detekuje. Na druhou stranu je zde šance cca 30%, že objekt nezachytí vůbec. Tento ukazatel se označuje jako miss rate. Ovšem když se podíváme na výsledky každého objektu zvlášť, tak zjistíme, že hodnoty jednotlivých objektů se od sebe dost odlišují. Například blok je svým tvarem a výřezy natolik složitý a nejednoznačný objekt, že se jej povedlo detekovat pouze párkrát. Naproti tomu objekty s primitivními hranami téměř vždy rozpoznal.



Obrázek 23: Celková úspěšnost detekce

Graf 23 reprezentuje celkovou úspěšnost detekce. Z dosažených výsledků zaznamenáváme, že rozdíly v použití jednotlivých variant jsou skutečně patrné. U jednotlivých variant jsme dosáhli následujících výsledků (viz tabulka 8). I v případě celkové úspěšnosti byla potvrzena původní hypotéza. Obě vylepšení společně skutečně přispívají k lepší a přesnější detekci. V případě diskriminativních hran a vylepšeného OCM dosáhl detektor nejvyšší úspěšnosti, která dosahuje 71,05%. Naopak bez použití jakýchkoliv vylepšení dosahuje detekce pouze 67,57% šance na úspěch.

Tabulka 8: F-score jednotlivých variant

Varianta	F - score
Originální hrany a výpočet vylepšeného OCM	67.55%
Výběr diskriminativnách hran a výpočet vylepšeného OCM	71.05%
Originální hrany a a výpočet klasického OCM	67.57%
Výběr diskriminativních hran a výpočet klasického OCM	70.02%

Je nutné vzít v potaz i výsledný čas algoritmu. S úspěšností 71.05% trvá detekce v jednom snímku při rozlišení 640×480 pixelů průměrně 6 vteřin. Časová složitost se odvíjí především od počtu objektů v obraze a počtu průchodů skenovacím oknem. Pokud je obraz příliš zahlcený objekty a náš detektor zrovna testuje objekty jednoduchých tvarů, tak první stupeň kaskády propustí příliš mnoho snímků. Proto druhá fáze, která je poměrně časově náročná, ale za to maximálně přesná, běží podstatně delší dobu. Výsledný čas běhu algoritmu by bylo možné ještě snížit, ovšem na úkor chybějící nebo falešné detekce. Dosažené výsledky a výsledný čas algoritmu je spojen s 10ti průchody skenovacího okna a odstupem referenčních bodů 3 pixely.

6 Závěr

Algoritmus pro detekci beztexturových 3D objektů je funkční a rychlý. Ovšem úspěšnost detekce závisí na typu rozpoznávaného objektu (složitosti tvaru, zaoblení, příliš mnoho zakřivení a detailů) a také na osvětlení scény. V případě velkého jasu není schopen algoritmus správně detekovan hrany. Ovšem v případě jednoduchých či specifických objektů dopadl výsledek podle očekávání. Celková úspěšnost algoritmu v nejlepším případě (tedy za použití obou vylepšení) dosahuje až 71,05%.

Výpočet příznaků vzdálenosti a orientace je pro rozpoznávání objektů dostačující. Detekce objektů a určení jejich přibližné polohy je velmi přesné. Ale jelikož se výpočet obou těchto příznaků odvíjí od detekce hran, tak je právě tento krok zásadní. A to je v případě různých osvětlení a hodnot jasu menší problém. Vektory příznaků jsou následně porovnávány na základě kvantovaných hodnot. Stejně jako je tomu v případě histogramů. Ale jelikož by bylo porovnání všech histogramů trénovacích snímků se všemi histogramy testovacích snímků časově neproveditelné, nahrazujeme tento krok vyhledáváním v indexovací tabulce. Každá buňka tabulky je jednoznačně identifikovaná vzdáleností, orientací a indexem referenčního bodu. Na pozici této buňky se pak nachází všechny trénovací snímky, které splňují podmínku vzdálenosti, orientace a indexu. Na základě těchto hodnot pak sbírají trénovací snímky hlasy. V případě velkého počtu hlasů postupují vybrané snímky do druhé fáze kaskádového modelu, kde dojde k přesnějšímu určení výsledné detekce. K tomu využíváme výpočet OCM. Tato metoda nám vrátí skupinu několika detekčních oken. Aby se výsledky nepřekrývaly, je nutné aplikovat metodu zvanou non-maxima suppression. Ta vybere ze skupiny detekčních oken ta nejpřesnější, vykreslí je do výsledného obrázku a ještě zařídí, aby se okna příliš nepřekrývala.

Segmentace obrazu a následná lokalizace objektů nám poskytují všechny důležité informace, které můžeme dále analyzovat. Ačkoliv si to mnozí z nás neuvědomují, tento princip nás již nyní provází v každodenním životě a je jisté, že v budoucnosti se s ním budeme setkávat stále častěji. Například v supermarketu lze odhadnout zralost a kvalitu ovoce a zeleniny umístěné na váhu na základě jejich barvy a tvaru. Váha se zabudovanou kamerou poté vyhodnotí konkrétní cenu pro zákazníka. V současné době se s podobným principem pro určení kvality a jakosti výrobků setkáváme pouze v průmyslu. Tento a spoustu dalších nápadů, jejichž základem je detekce a analýza obrazu sice ještě v reálném světě nepotkáte, ale v současné době se vyvíjí. Takže je pravděpodobné, že se s nimi za pár let setkáme.

Literatura

[1] Autor neznámý: Digitální obraz,
http://www.wikiskripta.eu/index.php/Digit%C3%A1ln%C3%AD_obraz

[2] Autor neznámý: Metody rozpoznání objektů v obraze,
http://www.fbmi.cvut.cz/files/predmety/3528/public/Metody%
20rozpozn%C3%A1n%C3%AD%20objekt%C5%AF%20v%20obrazu.pdf

[3] Zedník P.: Detection and Localization of Texture-Less Objects in RGB-D Images,
https://dspace.cvut.cz/bitstream/handle/10467/62026/F3-DP-2015-Zednik-
Pavel-thesis.pdf?sequence=3&isAllowed=y

[4] Autor neznámý: Cascade classification,
http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html

[5] Sojka E., Gaura J., Krumnikl M.: Matematické základy digitálního zpracování obrazu,
http://mrl.cs.vsb.cz/people/sojka/dzo/mzdzo.pdf

[6] Sojka E.: Digitální zpracování a analýza obrazu,
http://mrl.cs.vsb.cz/people/sojka/dzo/digitalni_zpracovani_obrazu.pdf

[7] Huttenlocher D.: Distance Transforms,
https://www.cs.cornell.edu/courses/cs664/2008sp/handouts/cs664-7-dtrans.pdf

[8] Hlaváč V.: Hledání hran,
http://cmp.felk.cvut.cz/cmp/courses/33DZOzima2006/slidy/detekce_hran.pdf

[9] Autor neznámý: Histograms,
https://statistics.laerd.com/statistical-guides/understanding-histograms.php

[10] Autor neznámý: The chamfer system,
http://www.gavrila.net/Research/Chamfer_System/chamfer_system.html

[11] Cai, H., Werner, T., Matas, J.: Fast Detection of Multiple Textureless 3-D Objects,
http://cmp.felk.cvut.cz/ matas/papers/cai-2013-textureless-icvs.pdf

[12] Autor neznámý: OpenCV,
http://opencv.org/

[13] Felsberg M.: Robot Vision Systems,
https://www.cvl.isy.liu.se/education/graduate/opencv/Lecture1_History.pdf

[14] Autor neznámý: Train-opt2,
http://cmp.felk.cvut.cz/data/textureless/CMP-8objs/train-opt2/

[15] Klos, D.: Počítání tlakových láhví v obraze,
https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=119345

[16] Autor neznámý: Test,
http://cmp.felk.cvut.cz/data/textureless/CMP-8objs/test/